IMPARIAMO A PROGRAMMARE IN BASIC CON IL PET™/CBM™

Rita Bonelli

GRUPPO EDITORIALE JACKSON

BASIC
BASIC
BASIC

IMPARIAMO A PROGRAMMARE

di
Rita Bonelli

GRUPPO
EDITORIALE
JACKSON
Via Rosellini, 12
20124 Milano

PREFAZIONE

Questo manuale è il secondo edito dalla Jackson scritto per mano della dr. Rita Bonelli. In precedenza la medesima autrice aveva redatto un agile e limpido libretto dal titolo: "Impariamo a programmare in BASIC con lo ZX80" che ha ottenuto presso il pubblico un successo notevole, superiore alle aspettative ma non - a parere di chi scrive - imprevedibile.
Come è stato messo in evidenza in quella prefazione, di cui questa - insieme al presente testo - è un poco il seguito ideale, oggi quello dei personal computer ha assunto un significato che non esitiamo a definire di carattere *sociale*. Una nuova e nuovissima fascia di utenti si affaccia all'orizzonte (affollandolo sempre più) dalla Scienza dell'Informazione: dirigenti di aziende minuscole e addirittura microscopiche, appassionati di elettronica e informatica, hobbisti e tanti, tantissimi giovani: anche quelli che non hanno la fortuna di poter seguire a scuola corsi di alcun genere sulla materia. Che il fenomeno abbia un carattere di massa nessuno può più metterlo in dubbio (ed è destinato ad accentuarsi, con la progressiva decrescita dei prezzi dell'hardware).
Non ci si deve neanche nascondere che tutto ciò possa avere delle connotazioni "consumistiche" (anche se certamente con carattere molto più nobile e dignitoso di altre frivolezze della società opulenta). Resta comunque il fatto che, da una potenziale vasta diffusione della cultura informatica, fino a non molto tempo fa confinata entro cerchie ristrette, per non dire élitarie, non può venire che del bene per una società più matura, ove la tecnologia si diffonde e si fa al tempo stesso strumento di cultura e di progresso. Ma ecco il nodo: l'hardware, senza quella cosa soffice, anzi diremmo "sottile" che gli anglofoni chiamano software, è materia inerte priva di afflato vitale. Di qui la fame di "logicale" da parte di tutta quella brava gente di cui sopra. Essa si estrinseca in termini spesso di ansiosa ricerca di prodotti software preconfezionati ma non è certamente disgiunta dall'esigenza di *imparare a fare da sé*, allo scopo di capirci qualcosa, familiarizzarsi con l'oggetto, vederci chiaro il più possibile.
Ecco allora trovata una chiave per la spiegazione del successo di libri come il precedente e, ne siamo sicuri, come questo: di più ampio respiro. Ma se esaurissimo il discorso in questi soli termini saremmo estremamente riduttivi. Infatti l'autrice, forte di una pluriennale esperienza professionale seguita da più recente attività didattica, è riuscita anche qui - anzi in una maggior misura -ad ottimizzare diverse esigenze: chiarezza e semplicità di esposizione (per venir incontro a chi muove i primi, sovente incerti passi in

questa materia che non è certo priva di aspetti ostici); abbondanza di esempi anche orientati praticamente; concretezza di riferimento. Vi si parla infatti di una ben precisa macchina, tra le più diffuse sul mercato.
Ma l'autrice non ha certamente fatto un manuale d'uso sia pure ben curato, ma un vero e proprio testo didattico. Invertendo la formula purtroppo troppo in voga in tanti ambienti scolastici malati di eccessivo sperimentalismo, vi si procede infatti in modo organicamente sistematico: dalle premesse di carattere generale (in primis: l'analisi di un problema da automatizzare in un algoritmo procedurale) si va avanti con l'illustrazione delle caratteristiche essenziali del BASIC in modo cioè che il "background" preceda la conoscenza e l'uso della macchina. Su questa materia poi, come s'è detto, non vi è certo avarizia di casi interessanti e di esercizi proposti e risolti.
Insomma vi sono tutti i numeri perché non solo si rinnovi un successo editoriale, ma venga dato un contributo alla crescita del mercato e della sua cultura. Ai fruitori di queste prodigiose macchinette l'autrice Bonelli rivolge un invito alla pazienza: al termine, chiunque abbia tale virtù può essere in grado se non di fare cose strepitose sicuramente di ottenere soddisfacenti risultati. Anzi, la dr. Bonelli, che pur ha lavorato per lunghi anni su sistemi mastodontici è convinta che queste piccole macchine non solo hanno potenzialità tutt'altro che trascurabili ma, nella didattica hanno una funzione vitale. Di qui l'auspicio di una sempre maggior diffusione di questi economici sistemi e della scienza informatica nel più vasto spazio di scolarità possibile: anche nella media inferiore (come concordano autorevoli esperti d'oltr'alpe che queste cose le hanno comprese prima di noi). Ebbene: quest'opera può dare un suo contributo verso tale traguardo.

Gianni Giaccaglini

SOMMARIO

CAPITOLO 5 - STRUTTURA DEL PET

CAPITOLO 6 - TRATTAMENTO DEI FILES

CAPITOLO 7 - L'USO DEI FLOPPY DISK

CAPITOLO 9 - SISTEMA AVANZATO DI PROGRAMMAZIONE CON IL DISCO

CAPITOLO 1

INTRODUZIONE

1.1. Prefazione

Scopo di questo manuale è quello di iniziare alla programmazione dei calcolatori elettronici le persone inesperte. Perchè lo scopo possa essere raggiunto il manuale, dopo una breve introduzione all'argomento, passa ad insegnare, per gradi, la programmazione in linguaggio BASIC, servendosi dei calcolatori PET della COMMODORE.

Gli argomenti sono svolti facendo riferimento alle prove pratiche, che sono essenziali per un buon apprendimento.

Nel Capitolo 10 si danno anche informazioni sulla programmazione in linguaggio macchina.

1.2. Struttura di un calcolatore elettronico

Il calcolatore elettronico è una macchina in grado di elaborare informazioni numeriche. Il nome completo è: calcolatore automatico elettronico digitale:

- calcolatore, perchè esegue dei calcoli;
- automatico, perchè, dopo averlo avviato ed aver predisposto i suoi programmi di lavoro, funziona automaticamente;
- elettronico, perchè realizzato usando le tecniche elettroniche;
- digitale, perchè, dall'inglese "digit", tratta cifre.

Abbiamo parlato di calcoli e di cifre; in realtà con il calcolatore elettronico si elaborano anche informazioni non di tipo numerico, solo che tali informazioni sono presentate al calcolatore in forma di codici numerici. Nella figura 1-1 è riportato lo schema generale di un calcolatore elettronico.

Le parti componenti sono:

- Unità Centrale (CPU);
- Unità di ingresso;
- Unità di uscita;
- Memoria Secondaria di massa.

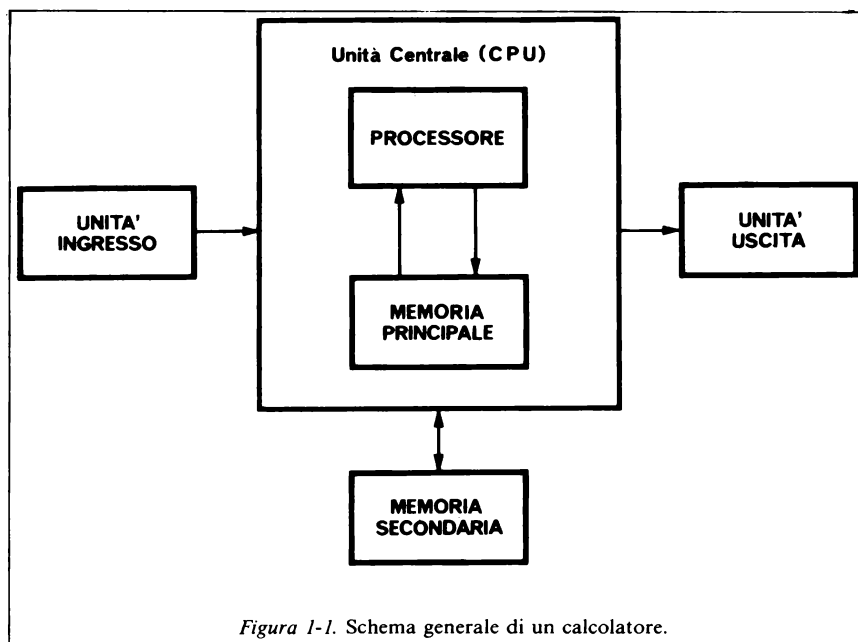



Figura 1-1. Schema generale di un calcolatore.

L'unità centrale è formata da:

- Un Processore che comprende:
  - Unità di controllo,
  - Unità Aritmetica Logica,
  - Registri Speciali.
- La Memoria che comprende:
  - Memoria ROM (a sola lettura)
  - Memoria RAM (per lettura e scrittura).

L'unità di ingresso generalmente è una tastiera. *L'unità di uscita* nei casi più comuni è un video o una stampante.

La *memoria secondaria* di massa è necessaria se si vogliono memorizzare grandi quantità di informazioni.

L'*unità di controllo* governa il funzionamento del sistema servendosi dei programmi, che sono memorizzati nella ROM o nella RAM, e dell'unità Aritmetico logica. Il *programma* è un insieme di istruzioni elementari che il calcolatore è in grado di eseguire; può essere stabilmente memorizzato nella ROM o temporaneamente memorizzato nella RAM. I programmi memorizzati nella ROM sono forniti dalla casa costruttrice del calcolatore elettro-

nico. La memoria può essere considerata come formata da un insieme di caselline, ognuna dotata di un indirizzo di riconoscimento, e nelle quali possono essere immagazzinate informazioni. Queste informazioni possono essere sia istruzioni di programma che dati su cui operare. Al livello più basso le informazioni sono memorizzate in forma binaria, cioè servendosi di un alfabeto a due simboli, detti bit, che possono avere o il valore zero o il valore 1. In conseguenza l'aritmetica del calcolatore è l'aritmetica binaria. Essa comunque è l'aritmetica "interna" del calcolatore, in quanto l'utente può tranquillamente trattare con il calcolatore servendosi dall'esterno della usuale aritmetica decimale.

Le caselline, di cui abbiamo parlato prima, possono in generale contenere una informazione di 8 bit e prendono il nome di *byte*. Le informazioni, codificate in modo opportuno, vengono memorizzate in gruppi di bytes.

Esiste una procedura automatica che consente all'unità di governo di andare a prelevare dalla memoria le istruzioni del programma una dopo l'altra e di eseguirle. Ogni calcolatore elettronico è dotato di un gruppo di istruzioni elementari che costituiscono il *"linguaggio macchina"*. Si possono quindi scrivere i programmi in linguaggio macchina, memorizzarli nella memoria del calcolatore ed avviare il procedimento di esecuzione automatica. Questi programmi in linguaggio macchina sono i migliori, se scritti da un programmatore molto esperto, perchè occupano poca memoria e sono molto veloci nei tempi di esecuzione. Solo che scrivere lunghi programmi in linguaggio macchina non è facile. Per questa ragione sono stati messi a punto dei linguaggi simbolici di programmazione di più facile apprendimento per l'uomo, formati da istruzioni che corrispondono a gruppi di istruzioni di linguaggio macchina. Nello stendere un programma in un linguaggio simbolico si devono rispettare rigorosamente le regole del linguaggio stesso. Il lavoro di traduzione in linguaggio macchina, viene svolto da un programma traduttore il quale controlla la grammatica e la sintassi del linguaggio che deve tradurre.

1.3. Introduzione alla programmazione in linguaggio BASIC

In questo corso si studierà il BASIC con l'ausilio di un piccolo calcolatore dotato di una tastiera per scrivere e di un video per leggere, oltre che di altre periferiche che vedremo più avanti.

BASIC significa: Beginners All-purpose Symbolic Instruction Code (Linguaggio di programmazione generale per principianti).

Il nostro calcolatore è dotato di un Sistema Operativo efficiente e di facile apprendimento, che studieremo durante il corso. Il *sistema operativo* è un insieme di programmi che permettono la gestione automatica delle risorse

Figura 1-2. Computer modello 3032 e unità floppy disk.

del sistema. Per iniziare ci basta dire che mediante una semplice procedura ci mettiamo in condizione di colloquiare in linguaggio BASIC con il calcolatore servendoci della tastiera e del video.

Il BASIC non è il linguaggio base del calcolatore, cioè non è il linguaggio macchina, ma noi non ce ne accorgiamo perchè il Sistema Operativo del nostro calcolatore è stato preparato in modo che l'utente possa comunicare direttamente in linguaggio BASIC.

Il linguaggio BASIC è *interattivo,* cioè durante la scrittura di un programma BASIC nella memoria del calcolatore, il Sistema Operativo interviene segnalando gli errori formali che riconosce, e che, pertanto, il programmatore può immediatamente correggere. Quando si lavora in BASIC su un calcolatore, nella memoria del calcolatore è presente un programma (che fa parte del Sistema Operativo) che si chiama Interprete BASIC. Per questa ragione si dice che il BASIC è un linguaggio *interpretativo,* cioè le frasi del linguaggio BASIC vengono interpretate dal Programma interprete prima di essere eseguite. Naturalmente l'utente non si accorge di questa fase di interpretazione (traduzione).

Coloro che hanno già pratica di altri linguaggi di programmazione, sanno che, per esempio in Cobol, Fortran e Assembler, la fase di traduzione del programma avviene prima della prove del programma. Inoltre, sanno anche, che con tali linguaggi, dopo aver trovato e corretto gli errori, si deve rifare la fase di traduzione. Un programma BASIC è formato da una serie di frasi, che impareremo ad usare, e che sono le istruzioni del linguaggio.

Prima di poter scrivere un programma è necessario procedere allo studio del problema che si vuole risolvere programmandolo in BASIC, nel nostro caso. Schematizzando, possiamo dire che un programma opera sempre una trasformazione di dati; ci sono dei dati di ingresso (*input*), che devono essere elaborati per produrre dei dati di uscita (*output*). "Essere elaborati" significa che si devono trovare delle regole di calcolo (delle formule di calcolo o, come si usa dire, un *algoritmo)* che permettano di produrre i risultati richiesti senza errori e nel modo più veloce possibile.

Nei prossimi capitoli ci occuperemo dello studio dei problemi e dello studio del linguaggio Basic.

CAPITOLO 2

ANALISI DEI PROBLEMI - ALGORITMI - DIAGRAMMI A BLOCCHI

2.1. Analisi del problema e ricerca del metodo di soluzione

Il problema da risolvere deve essere ben posto, cioè deve essere chiaro cosa si vuole fare, da quali dati iniziali si parte e quali risultati si vogliono ottenere. Le caratteristiche dei dati di partenza devono essere definite, per esempio, se sono numeri interi o decimali. Analogamente devono essere definite le caratteristiche dei dati in uscita.

Una volta chiarito "cosa si vuol fare", bisogna trovare "come farlo", cioè ricercare il metodo di soluzione, in altre parole trovare l'algoritmo risolutivo. Facciamo un semplice esempio. Si debba risolvere questo problema:

TROVARE LA MEDIA ARITMETICA DI DUE NUMERI INTERI E CALCOLARLA CON UNA CIFRA DECIMALE.

Il problema è ben posto perchè ci dice che:

- i dati di input sono due numeri interi;
- il dato di output è la loro media aritmetica calcolata con una cifra decimale;
- il "cosa si vuol fare" è calcolare una media aritmetica tra due numeri interi.

Dovremmo a questo punto ricercare l'algoritmo risolutivo; non è molto complicato, si tratta di un concetto usuale, per calcolare la media di due numeri, basta fare la loro somma e poi dividerla per due. Chiamiamo A e B i due numeri, chiamiamo M la media, il nostro algoritmo scritto in formula di calcolo è:

$$M = \frac{A + B}{2}$$

Però dobbiamo ottenere che M sia calcolata con una cifra decimale.

Pensiamo ora a quale serie di istruzioni dobbiamo dare al calcolatore perchè possa risolvere questo problema.

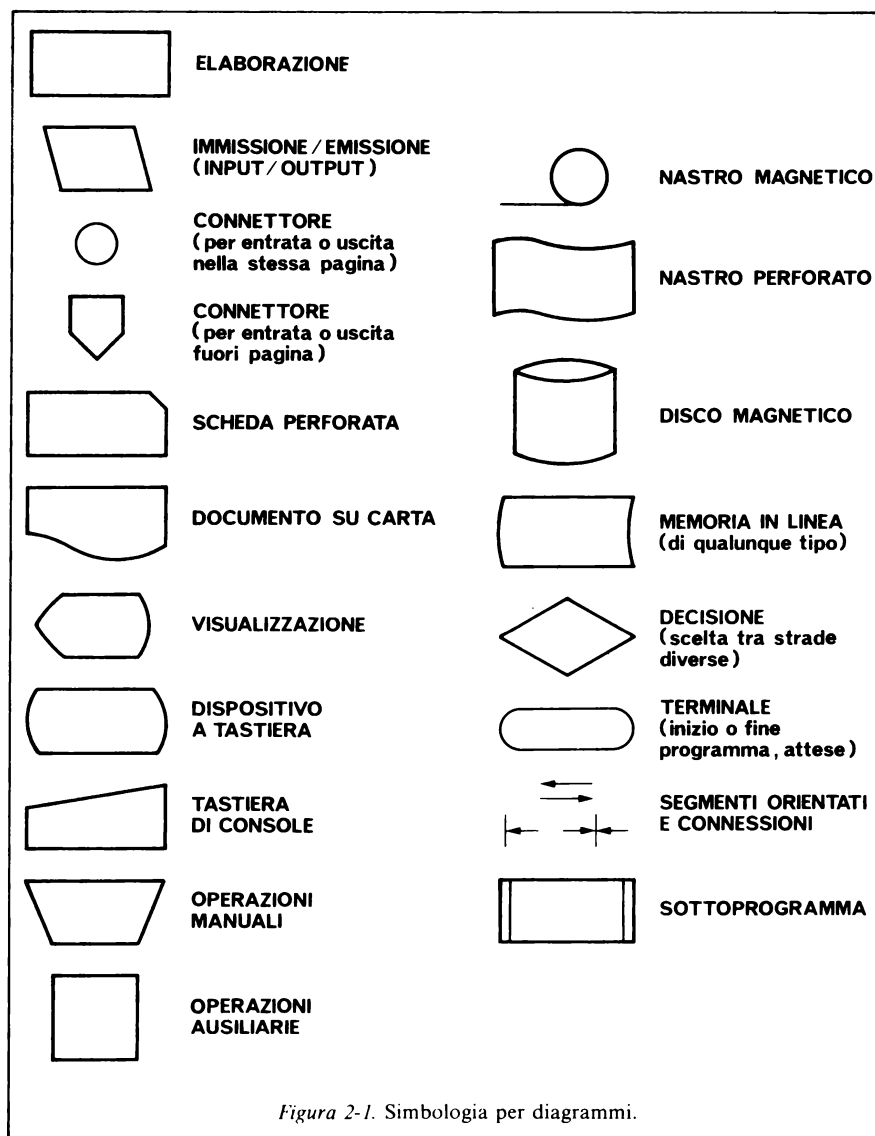



Figura 2-1. Simbologia per diagrammi.

La serie di istruzioni da dare può essere la seguente:

1. Leggi dall'esterno il numero (A) e memorizzalo;
2. Leggi dall'esterno il numero (B) e memorizzalo;
3. Calcola $M = (A + B)/2$ con una cifra decimale (/, sta per diviso);
4. Invia all'esterno la risposta M.

Se siamo capaci di far eseguire al calcolatore questa serie di istruzioni (programma), abbiamo risolto il problema di calcolare la media aritmetica di due qualsivoglia numeri interi.

A questo punto potremo passare rapidamente alla *stesura o codifica* del programma per risolvere il problema di cui abbiamo svolto l'analisi e descritto l'algoritmo risolutivo.

Si usa anche descrivere gli algoritmi risolutivi servendosi della tecnica dei *diagrammi a blocchi.* I *diagrammi a blocchi* sono la descrizione grafica degli algoritmi.

2.2. Diagrammi di flusso e diagrammi a blocchi

Abbiamo detto che un programma opera una trasformazione di dati, abbiamo, cioè, un flusso di informazioni che entrano nel calcolatore, vengono elaborate e producono un flusso di informazioni che escono. Il flusso di informazioni è costituito da diversi documenti che provengono da varie fonti o che sono destinati ad altre fonti.

Si usa ricorrere ad una rappresentazione grafica per descrivere il flusso completo delle informazioni che si elaborano e questa rappresentazione grafica prende il nome di *diagramma di flusso.* Il *diagramma di flusso* rappresenta il flusso delle informazioni all'esterno del calcolatore. Il *diagramma a blocchi,* rappresenta invece il flusso delle informazioni all'interno del calcolatore, cioè le attività del programma.
Per questi due tipi di diagrammi si usa una simbologia grafica che descriveremo brevemente e che risulta molto significativa. Si usano delle figure geometriche con all'interno delle scritte esplicative, collegate tra loro da segmenti orientati (con frecce).

Negli esempi iniziali riportati in questo manuale vale sempre un diagramma di flusso molto semplice, che tracciamo qui.

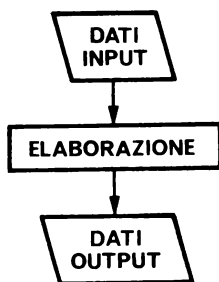
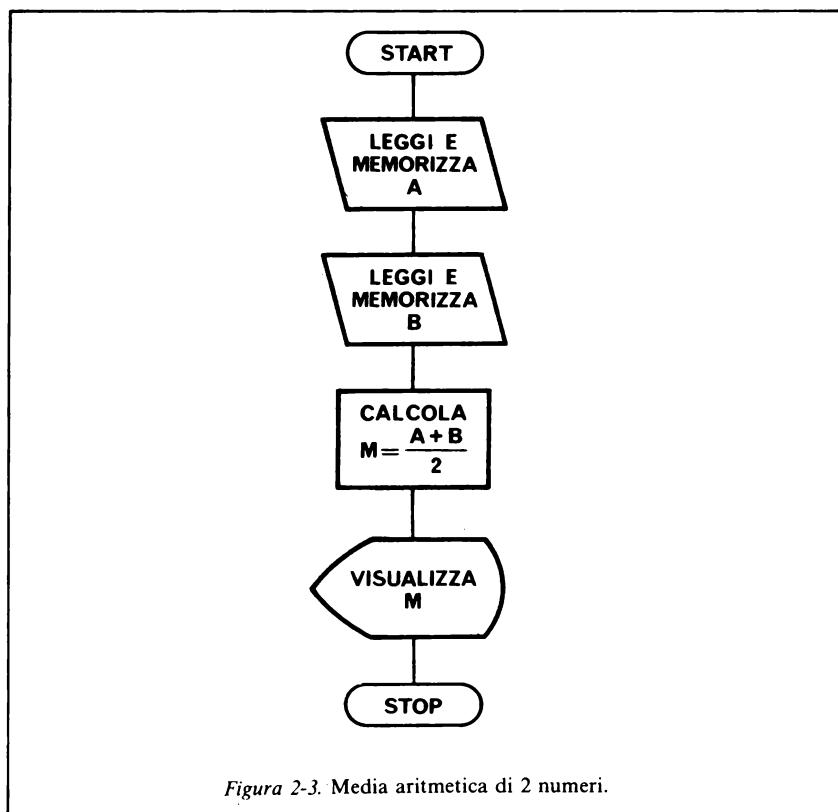



Figura 2-2. Schema generale di flusso.

Infatti siamo quasi sempre nelle condizioni di:

- dati di ingresso;
- elaborazione dei dati;
- estrazione dei risultati

Quando si affronteranno problemi più complessi, si vedrà che, tracciare all'inizio il diagramma di flusso della procedura da affrontare, rende più semplice il lavoro ed aiuta a non fare errori. Tracciamo qui il diagramma a blocchi dell'algoritmo visto nel paragrafo precedente per il calcolo della media di due numeri.

Figura 2-3. Media aritmetica di 2 numeri.

2.3. Calcolo della media aritmetica di 10 numeri interi

Il problema sia:

TROVARE LA MEDIA ARITMETICA DI 10 NUMERI INTERI E CALCOLARLA CON UNA CIFRA DECIMALE.

È ovvio che potrebbe essere applicato l'algoritmo già visto leggendo e memorizzando 10 numeri invece di due, facendo la somma dei 10 numeri e poi dividendola per 10. Si vede però che si ripeterebbe 10 volte la stessa operazione e risulterebbe un pò noioso tracciare, per esempio, il relativo diagramma a blocchi. Siamo di fronte ad un gruppo di operazioni ripetitive o *cicliche;* tale situazione è molto comune nella programmazione, vediamo dunque come può essere affrontata.

Invece di chiamare i 10 numeri: A, B, C, D, E, F, G, H, I, L, oppure N1, N2, N3, N4, N5, N6, N7, N8, N9, N10 chiamiamo globalmente i 10 numeri con il nome N, facendo seguire N da una coppia di parentesi che contengono un numero, chiamato indice, che varia da 1 a 10 Così:

N(1), N(2), N(3), N(4), N(5), N(6), N(7), N(8), N(9), N(10).

Per rappresentare sinteticamente i dieci numeri scriviamo:
N(K), dove K è una variabile usata come indice che può variare da 1 a 10:

quando K = 1,N(K) rappresenta il primo dei 10 numeri;
quando K = 5,N(K) rappresenta il quinto dei 10 numeri;
quando K = 10,N(K) rappresenta il decimo dei 10 numeri.

Introduciamo ora il concetto di contatore di ciclo. Il *contatore di ciclo* è una variabile (contenitore che deve essere riempito e quindi può assumere diversi valori nel tempo) che serve per contare il verificarsi di una situazione, per esempio contare da 1 a 10 mentre leggiamo 10 numeri. Introduciamo, anche, il concetto di *operazione logica di confronto,* cioè di una operazione che ci permette di verificare se il contatore di ciclo ha raggiunto un certo valore, ed in base a tale verifica scegliere la strada da percorrere.

Descriviamo verbalmente l'algoritmo risolutivo del problema posto:

1. chiamo K il contatore di ciclo e lo pongo al valore iniziale 1 : K = 1;
2. leggo dall'esterno un numero e lo memorizzo in N(K), N è il nome usato per l'insieme dei 10 numeri da trattare;
3. verifico se K è uguale a 10 (ha già letto il decimo numero), se K = 10 proseguo dal punto 5., se no, proseguo dal punto 4.;
4. aggiungo 1 a K passando al valore successivo e proseguo dal punto 2.;
5. ho terminato di leggere i 10 numeri e questi sono memorizzati in N(K) con K = 1,2, ..., 9,10. Pongo al valore zero la variabile M, dove voglio memorizzare la media. Pongo di nuovo al valore 1 il contatore, K = 1, per controllare il ciclo di somma.
6. aggiungo ad M il numero N(K);
7. verifico se K è uguale a 10 (ho sommato il decimo numero), se K = 10 proseguo dal punto 9. se no proseguo dal punto 8.;

8. aggiungo 1 a K passando al valore successivo e proseguo dal punto 6.;
9. calcolo la media M dividendo M per 10;
10. visualizzo il risultato, cioè M.

L'algoritmo descritto ha la caratteristica di memorizzare separatamente i 10 numeri letti in N(K); e questo non era precisamente richiesto dal problema; si chiedeva solo la media. Nel punto 5. abbiamo posto il valore iniziale zero in M, altrimenti la somma poteva essere errata. Vediamo il diagramma a blocchi dell'algoritmo descritto.

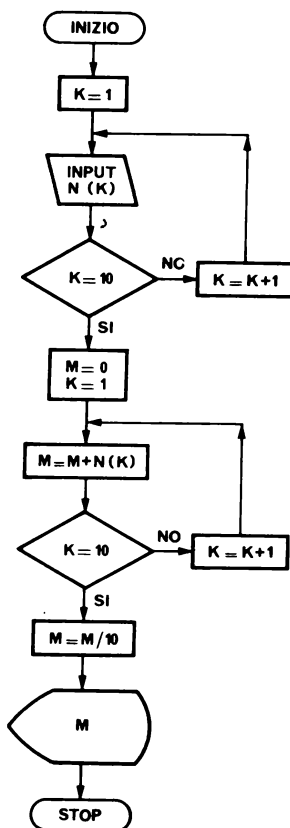



Figura 2-4. Media aritmetica di 10 numeri con memorizzazone di dati.

Risolviamo, il problema posto prima, con un altro algoritmo, più semplice del precedente; rinunciamo cioè a conservare nella memoria del calcolatore i dieci numeri. In questo caso, dopo avere inizialmente azzerato la variabile M, avremo un ciclo, percorso 10 volte, nel quale: si legge un numero e lo si somma in M. Alla fine del ciclo si calcola la media aritmetica. Passiamo alla descrizione verbale di questo algoritmo e ne diamo subito dopo il diagramma a blocchi.

1. Pongo la variabile M al valore zero;
2. pongo la variabile K, contatore del ciclo, al valore 1;
3. leggo dall'esterno il numero N (cioè memorizzo nella variabile N il numero letto; N questa volta non rappresenta un gruppo di variabili, ma una variabile singola);
4. sommo a M il numero N: $M = M + N$;
5. verifico se K è uguale a 10; se $K = 10$ proseguo dal punto 7., se no proseguo dal punto 6.;
6. aggiungo 1 a K per controllare il ciclo, e proseguo dal punto 3.;
7. calcolo la media M dividendo M per 10;
8. visualizzo il risultato, cioè M.

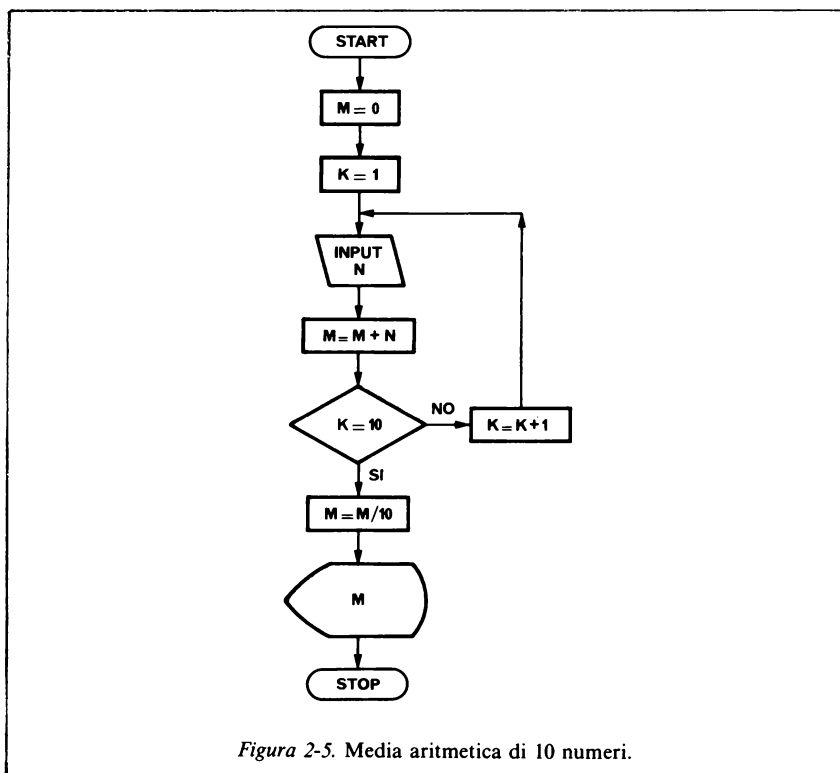



Figura 2-5. Media aritmetica di 10 numeri.

Nelle pagine precedenti abbiamo risolto un problema con due diversi procedimenti; in ambedue i casi calcoliamo la media aritmetica di 10 numeri e la visualizziamo, ma nel primo caso i numeri letti restano memorizzati nella memoria del calcolatore, mentre nel secondo caso no. Nel primo caso il contatore del ciclo K viene adoperato anche come indice della variabile N(K), che prende proprio il nome di *variabile con indice,* mentre nel secondo caso K serve solo per controllare il ciclo. Vogliamo riassumere quali sono le operazioni che si devono fare per potersi servire di un contatore in un algoritmo. Abbiamo visto che:

1. si pone un valore iniziale nel contatore, (nel nostro caso abbiamo inizializzato con il valore 1);
2. si controlla se il contatore è arrivato al valore finale; se il valore finale è stato raggiunto si abbandona il ciclo, se no si prosegue;
3. si incrementa il valore del contatore di 1 e si ripete il ciclo.

Queste tre operazioni si chiamano:

- inizializzazione,
- controllo,
- incremento,

e sono necessarie per gestire un contatore.

Tra il punto 1. ed il punto 2. di cui sopra, si svolgeranno le operazioni cicliche inerenti alla risoluzione del problema che si sta trattando. Naturalmente un contatore di ciclo può essere usato anche in altro modo e cioè partendo da un valore iniziale grande (nei nostri esempi, 10), controllando se il contatore è arrivato al valore 1, e se no, decrementando il contatore, cioè sottraendo il valore 1. È molto importante imparare a gestire bene un contatore per poter esporre con sicurezza algoritmi nei quali siano necessari dei cicli.

CAPITOLO 3

LE FRASI DEL LINGUAGGIO BASIC E L'USO ELEMENTARE DEL PET/CBM

3.1. Le frasi BASIC necessarie per codificare i programmi esempio del Capitolo 2

In BASIC ogni frase o istruzione è formata da tre parti:

1. il numero di linea che può variare tra 0 e 63999;
2. le direttive BASIC (parole chiave del linguaggio);
3. i parametri associati alla direttiva.

Il *numero di linea* serve al sistema per tenere in ordine le istruzioni del programma, cioè il programma viene eseguito per numeri di linea crescenti. È buona norma non usare numeri consecutivi per numerare le linee, ma saltare, per esempio di 10 in 10 così: 10, 20, 30, ecc. In tale modo, qualora si volesse modificare un programma, si possono inserire frasi numerandole opportunamente senza dover modificare quelle già scritte.

Le *direttive BASIC* verranno studiate una per volta chiarendone il significato.

I *parametri* servono per definire l'azione che il calcolatore deve compiere e possono essere di diverso tipo. Tra essi citiamo ora le *costanti* e le *variabili.* Le *costanti* sono o numeri o insiemi di caratteri. Le *variabili* sono "Tasche vuote che aspettano di essere riempite"; cioè sono dei nomi simbolici che inventa il programmatore e che servono a contenere qualcosa che può venire ricevuto dall'esterno o prodotto dal programma nel calcolatore. Nel BASIC distinguiamo quattro tipi di variabili; esse sono:

- variabili intere,
- variabili decimali,
- variabili stringa,
- variabili logiche (delle quali si parlerà nel seguito).

Le regole per la formazione dei nomi delle variabili sono le seguenti:

- le *variabili intere* hanno un nome che può essere formato da uno o due caratteri seguiti dal carattere %. Il primo carattere deve essere alfabetico, il secondo può essere numerico o alfabetico. Il %

significa che la variabile contiene un numero intero. Il valore della variabile intera varia da —32767 a + 32767; cioè la variabile intera non può contenere un numero fuori da questo intervallo di valori. Esempio: AA%, B2%, CC%, X5%, ecc.;

- le *variabili decimali* hanno un nome che può essere formato da uno o due caratteri. Il primo carattere deve essere alfabetico mentre il secondo può essere numerico o alfabetico. Le variabili decimali consentono una precisione di 9 cifre significative. Esempio: AA, A5, AX, X9, GG, ecc.;
- le *variabili stringa* hanno un nome che può essere formato da uno o due caratteri seguiti dal carattere $. Il primo carattere deve essere alfabetico, il secondo può essere alfabetico o numerico. Il carattere $ significa che la variabile contiene un insieme di caratteri alfanumerici. La variabile stringa è formata al massimo da 80 caratteri, cioè può contenere al massimo 80 caratteri. Esempio: A3$, CV$, U8$, ecc..

Abbiamo elencato i tre tipi di variabili considerandole come variabili singole. Ogni tipo di variabile già descritta diventa variabile con indice se si fa seguire il nome della variabile da una coppia di parentesi che contenga uno o più indici, separati da virgole. Per il momento ci limitiamo a considerare le variabili con un solo indice. Avremo quindi, per esempio:

- A3% (K) per variabile intera con indice,
- AA(K) per variabile decimale con indice,
- B4$(K) per variabile stringa con indice.

Le variabili con un solo indice prendono usualmente il nome di *vettori* oppure *matrici a una dimensione* oppure *tabelle monodimensionali.* Il numero totale di dimensioni (indice) non può superare 255 ed il numero totale di byte occupati non può superare 32767 (si veda Appendice C). Si può quindi affermare che, compatibilmente con la memoria disponibile, non si hanno limiti nel dimensionamento delle variabili con indice.

In questa prima parte dello studio del linguaggio BASIC faremo riferimento al PET dotato di *tastiera* per l'ingresso dei dati e di *video* per la visualizzazione (uscita dei dati).

Passiamo a descrivere le frasi del linguaggio che ci consentano di codificare i programmi esempio visti nel capitolo precedente.

ISTRUZIONE INPUT per l'ingresso dei dati della tastiera. Il formato è:

INPUT lista di variabili separate da virgole.

Esempio: INPUT A, B, C pone il calcolatore in attesa di leggere tre dati numerici decimali dalla tastiera; questi vengono memorizzati rispettivamente nelle 3 variabili A, B, C.

Il calcolatore, quando incontra questa istruzione, visualizza al video un punto interrogativo (?) ed il programma si ferma in attesa che venga soddisfatta la richiesta di dati. Se l'operatore risponde con dati in numero inferiore a quelli richiesti, il BASIC accetta i dati ma ripropone due punti interrogativi (??) e resta in attesa del completamento dei dati. I dati devono essere separati da una virgola come nella lista di richiesta. Se l'operatore scrive più dati di quelli richiesti (nell'esempio: 5.1, 4.56, 7.8, 9.23 sono 4 dati invece di 3) il BASIC risponde ?EXTRA IGNORED ed accetta per buoni quelli che soddisfano la richiesta. Per fare accettare i dati al calcolatore, l'operatore deve schiacciare il tasto RETURN. Quando il calcolatore visualizza il ? al video, viene attivato il programma di *correzione dello schermo* mentre lampeggia il cursore in attesa dei dati. L'operatore può sfruttare tutte le possibilità del programma di correzione dello schermo fino a quando schiaccia il tasto RETURN. In questo momento i dati vengono consegnati al BASIC. Durante l'attesa dei dati, non viene sentito il tasto di STOP. Se la richiesta è per una stringa, i caratteri della stringa devono esser scritti così: "STRINGA" cioè tra apici, se si desidera iniziare o terminare la stringa con degli spazi. Negli altri casi si scrive la sequenza di caratteri senza apici. Si deve ricordare che, se non si usano gli apici come delimitatori, viene accettato come primo carattere significativo il primo carattere diverso da blank. Se ad una richiesta di dati si risponde solo con il tasto RETURN, il BASIC considera terminato il programma e scrive READY; se si scrive CONT il programma continua. Se la richiesta è per una stringa, la risposta "" (stringa nulla) viene accettata.

Si può scrivere in modo più significativo la frase INPUT, facendo uso di una stringa tra apici prima della richiesta dei dati, così:

10 INPUT "DATA DI NASCITA"; A

viene visualizzato: DATA DI NASCITA? e l'operatore risponde con la data e poi RETURN. Se la richiesta è per un tipo di dati e l'operatore sbaglia, la risposta del calcolatore è:

?REDO FROM START

e l'operatore risponderà di nuovo.

ISTRUZIONE LET, per assegnare un valore ad una variabile, sia esso una costante o il risultato del calcolo di una espressione. Il formato è:

LET variabile = espressione

dove: *variabile* è il nome di una variabile,
espressione è qualunque espressione aritmetica e quindi nella sua forma più semplice una costante. Esempio:

A1 = 18 — assegna il valore 18 ad A1 oppure
LET A1 = 18

Le due frasi scritte sono equivalenti perché *la parola chiave LET può anche essere omessa.* Esempio: BB = B1 + 17.6 – somma il valore di B1 a 17.6 e mette il risultato in BB, oppure

LET BB = B1 + 17.6

Nel primo esempio è stato assegnato alla variabile decimale A1 il valore intero 18; il sistema lo accetta e lo memorizza nella forma 18.0.

Il BASIC elabora qualunque espressione scritta usando variabili, costanti e gli opeatori BASIC, che sono:

+ serve per sommare,
— serve per sottrarre,
* (asterisco) serve per moltiplicare,
/ serve per dividere,
↑ (freccia in su) serve per elevare a potenza,
= ha il normale significato di uguale,
(parentesi aperta
) parentesi chiusa.

Le espressioni si scrivono con le normali regole della matematica, usando le parentesi a coppie per racchiudere operazioni da calcolare con precedenza. Si possono usare diverse coppie di parentesi, facendo attenzione a non lasciarne di aperte.

L'ordine di esecuzione delle operazioni è il seguente:

- si procede da sinistra a destra;
- si eseguono dapprima le operazioni racchiuse tra le parentesi più interne;
- si eseguono prima gli elevamenti a potenza;
- seguono le moltiplicazioni;
- seguono le divisioni;
- seguono le somme e le sottrazioni nell'ordine in cui sono scritte.

I calcoli vengono portati avanti tenendo conto del tipo di variabili coinvolte. Il valore dell'espressione calcolata viene assegnato alla variabile che sta a sinistra dell'uguale. Se tale variabile è di tipo intero, i decimali eventualmente generati dal calcolo vengono persi. Esempio:

X = (—B + (B↑2 —4 * A * C) ↑ 0.5) / (2 * A)

viene calcolata così:

- $[B^2 - 4 \times A \times C]$
- viene estratta la radice dal valore trovato,
- viene sommato il valore trovato a —B,
- viene calcolato 2 x A,
- viene diviso il penultimo valore trovato per l'ultimo.

ISTRUZIONE PRINT, per scrivere sullo schermo. Il formato è:

PRINT lista di variabili separate da virgole o da punti e virgola.

Esempio: PRINT M - fa apparire il valore di M sullo schermo a partire dalla posizione attuale del cursore luminoso. Esempio: PRINT A, B - fa apparire il valore di A sullo schermo a partire dalla posizione attuale del cursore, poi spazia per effetto della virgola e fa apparire il valore di B. Esempio PRINT A;B fa apparire il valore di A sullo schermo a partire dalla posizione attuale del cursore e poi fa uno spazio e fa apparire il valore di B, cioè il punto e virgola non fa fare la spaziatura lunga.

Nei tre esempi visti, dopo aver stampato, il cursore va a capo. Se non si vuole andare a capo, si deve far terminare la lista di variabili con un punto e virgola o con una virgola. Esempio: PRINT A, B; - stampa A, salta e stampa B e non va a capo.

Riepiloghiamo le regole per l'uso della virgola e del punto e virgola come separatori delle variabili da stampare. Il *punto e virgola* come *separatore* opera così:

- fa stampare i dati uno di seguito all'altro nel loro *formato di stampa;*
- non fa andare a capo se viene posto dopo l'ultima variabile della lista.

La *virgola* come *separatore* opera così:

- attiva la tabulazione automatica di 10 in 10 dei dati sullo schermo con le seguenti modalità:
  - se il dato occupa più di 7 caratteri, spazia di (10 - il numero di caratteri) + 10 prima del prossimo dato;
  - se il dato ocupa meno di 7 caratteri o 7 caratteri spazia di (10 - il numero di caratteri) prima del prossimo dato.

Vediamo ora qual'è il *formato di stampa* dei dati in BASIC.

I *numeri interi o decimali* vengono stampati:

- se sono positivi, facendoli precedere e seguire da uno spazio, così un numero di 3 cifre occupa 5 posizioni sul video;
- se sono negativi, facendoli precedere dal segno - e seguire da uno spazio, così il numero − 453 occupa 5 posizioni di stampa.

Le *stringhe alfanumeriche* vengono stampate come sono senza aggiungere spazi. Per esempio la frase:

PRINT "OGGI PIOVE"; "DOMANI NON SI SA"

viene visualizzata così: OGGI PIOVEDOMANI NON SI SA, infatti il punto e virgola non fa spaziare. Invece la frase:

PRINT "OGGI PIOVE", "DOMANI NON SI SA"

viene visualizzata così: OGGI PIOVE   DOMANI NON SI SA, infatti la prima stringa occupa 10 posizioni, poi spazia di 10 e scrive la seconda stringa.

Per quanto riguarda *la stampa dei numeri* dobbiamo aggiungere alcune considerazioni. Per i numeri *interi* abbiamo già visto che si ha una limitazione nella grandezza; infatti il numero intero deve essere compreso tra —32767 e + 32767. Comunque i numeri interi vengono stampati senza problemi. Per i numeri *decimali* invece il calcolatore controlla se il loro valore assoluto è compreso tra 0,01 e 999999999,2 se lo è, il numero viene stampato con il punto decimale (ricordiamo che in notazione anglosassone si usa il punto decimale e non la virgola). Se il numero ha valore assoluto non compreso nell'intervallo citato, allora esso viene stampato in notazione scientifica. Esempi:

il numero 0,0034 viene stampato 3.4E-03 che significa
3.4 moltiplicato 10^{-3};
il numero − 1234567890,5 viene stampato − 1.2345678E+09 che significa
− 1.2345678 moltiplicato 10^{9}.

Fino ad ora abbiamo detto che la visualizzazione avviene a partire dalla posizione del cursore sullo schermo; in realtà si può modificare a programma la posizione del cursore sullo schermo usando nella frase PRINT una stringa che contenga i caratteri alfanumerici di controllo del cursore; tali caratteri sono:

- il comando di cancellazione dello schermo (SHIFT & CLEAR);
- il comando di capo pagina (HOME);
- il comando di spostamento a destra (CRSR);
- il comando di spostamento a sinistra (SHIFT & CRSR);
- il comando di spostamento verso il basso (CRSR);
- il comando di spostamento verso l'alto (SHIFT & CRSR).

Le dimensioni dello schermo video sono di 25 linee per 40 caratteri, tuttavia si possono stampare linee di 80 caratteri ed il BASIC fa andare automaticamente a capo dopo 40 caratteri. Il video può contenere in tutto 1000 caratteri, quando si superano i 1000 caratteri il BASIC fa scorrere automaticamente i dati verso l'alto, aggiungendo le nuove linee.

ISTRUZIONE STOP: per interrompere l'esecuzione di un programma. Quando il programma incontra questa istruzione, l'esecuzione viene interrotta e viene restituito il controllo al BASIC che stampa READY dopo aver stampato BREAK IN LINE XXXX, dove XXXX è il numero della linea; se si vuole continuare, scrivere CONT e poi premere RETURN.
Se si vuole far ricominciare lo svolgimento del programma, si deve scrivere RUN e premere RETURN.

ISTRUZIONE END: per interrompere l'esecuzione di un programma con possibilità di ripresa. Il programma si ferma e non viene stampato nessun messaggio, cioè non viene restituito il controllo al BASIC. Se si preme il tasto CONT il programma prosegue con la frase successiva. Si può usare END dovunque in un programma.

ISTRUZIONE REM: per inserire commenti nel programma. Tale frase serve per rendere più chiaro un programma inserendo in qualunque punto delle frasi di commento. Tali frasi non sono esecutive. La frase REM termina alla fine della riga. Se il commento è lungo si usano frasi REM consecutive.

ISTRUZIONE DIM: per assegnare una dimensione alle variabili con indice. Il formato è:

DIM variabile (dimensioni)

dove variabile è il nome di una variabile di tipo intero, decimale o stringa e dimensioni è un numero intero. Se il programmatore omette la frase DIM in un programma e poi fa uso di una variabile con indice, tale variabile viene implicitamente considerata dimensionata a 10. Dal momento che per il BASIC gli indici iniziano da 0, una variabile di dimensione 10 ha in realtà 11 elementi, cioè l'indice varia da 0 a 10. Molto spesso il programmatore non usa l'indice 0, in tale caso si ha solo spreco di memoria.

ISTRUZIONE GOTO per far proseguire il programma da un determinato numero di linea. Il formato è:

GOTO xxxx

dove xxxx è il numero di una linea di programma. Quando il programma incontra il GOTO, salta alla linea indicata e prosegue in sequenza.

ISTRUZIONE IF per operare un controllo logico ed una scelta tra due diverse alternative. Per questa istruzione esistono 3 diversi formati:

1. IF condizione THEN istruzione

   se la condizione è verificata viene eseguita l'istruzione dopo THEN e poi prosegue con la linea seguente; se la condizione non è verificata, prosegue con la linea seguente. Esempio:
   40 IF A$ = "VERO" THEN PRINT "VERO"
   50 PRINT "ANALIZZATA CONDIZIONE"
   se A$ contiene "VERO" il calcolatore visualizza
   VERO
   ANALIZZATA CONDIZIONE
   se A$ non contiene "VERO", il calcolatore visualizza solo
   ANALIZZATA CONDIZIONE.
2. IF condizione GOTO numero di linea

se la condizione è vera prosegue dal numero di linea dopo GOTO; se la condizione è falsa prosegue dalla linea seguente. Esempio:

```
100 IF A% = 5 GOTO 300
110 PRINT "CONDIZIONE NON VERA"
......
...... STOP
300 PRINT "CONDIZIONE VERA"
```

3. IF condizione THEN numero di linea

   se la condizione è vera prosegue dal numero di linea dopo il THEN; se la condizione è falsa prosegue dalla linea seguente. Esempio:

```
100 IF A% = 5 THEN 300
110 PRINT "CONDIZIONE NON VERA"
......
...... STOP
300 PRINT "CONDIZIONE VERA"
```

Il formato 2 ed il formato 3 sono sostanzialmente equivalenti. Le condizioni da analizzare vengono scritte come due espressioni aritmetiche separate da un *operatore relazionale.* Gli operatori relazionali che il BASIC consente sono i seguenti:

= per uguale
> per maggiore di
< per minore di
<> non uguale
> =maggiore o uguale di
< =minore o uguale di.

La condizione analizzata in una frase IF fa nascere una *variabile di tipo logico,* senza nome simbolico, il cui valore è —1, se la condizione è vera, ed è 0 se la condizione è falsa. Gli operatori relazionali possono essere usati anche in espressioni aritmetiche; in questo caso viene usato, al posto dall'espressione relazionale, il valore della variabile logica generata. Esempio: LET X = Y + Z − (A > 5)

se A > 5 sottrae —1 e quinsi somma 1
se A ⩽ 5 sottrae zero e quindi non varia il valore precedentemente calcolato.

3.2. Codifica BASIC dei programmi esempio del Capitolo 2

Nelle pagine seguenti riportiamo la codifica in BASIC dei tre programmi esempio studiati nel Capitolo 2 alle pagine 7, 10 e 13. Possiamo fare i seguenti commenti:

- in tutti i 3 programmi si è terminato con le due frasi STOP e END; ne bastava una sola. Provate i programmi così e poi provateli togliendo la frase STOP e vedete la differenza;

- negli ultimi due programmi si è dimensionata la variabile N a 10, ma non si è fatto uso della posizione di indice 0, quindi c'è spreco di memoria. Provate a modificare il programma usando anche la posizione di indice 0 e quindi usando 11 numeri invece di 10; ma attenzione a fare tutte le variazioni necessarie!
- desideriamo mettere bene in evidenza frasi come:

$$K = K + 1$$

dato che questo tipo di scrittura, che sarebbe errato in una espressione aritmetica normale, è invece lecito in programmazione. Abbiamo detto che le variabili sono "tasche che devono essere riempite", ora se abbiamo in una tasca 5 caramelle e ne aggiungiamo un'altra, la tasca contiene 6 caramelle. Quindi la scritta K = K + 1 è pienamente accettabile in programmazione.

Dopo aver studiato la codifica dei tre programmi e dopo aver studiato il prossimo paragrafo che dà le prime norme operative per il PET, fate sul calcolatore tutte le prove necessarie per comprendere bene la parte già studiata.

Codifica BASIC del programma di pag. 7:

```
10 REM CALCOLO MEDIA ARITMETICA DI DUE NUMERI
20 INPUT"SCRIVI NUMERO A";A%
30 INPUT"SCRIVI NUMERO B";B%
40 LET M=(A%+B%)/2
50 PRINT M
60 STOP
70 END
```

I due dati di input A% e B% sono interi; la media M è un numero decimale.

Codifica BASIC del programma di pag. 10:

```
10 REM CALCOLO DELLA MEDIA ARITMETICA DI 10 NUMERI
20 REM CON MEMORIZZAZIONE NEL VETTORE N(K)
30 DIM N(10)
40 LET K=1
50 INPUT"SCRIVI UN NUMERO";N(K)
60 IF K=10 GOTO 90
70 LET K=K+1
80 GOTO 50
90 LET M=0
100 LET K=1
110 LET M=M+N(K)
120 IF K=10 GOTO 150
130 K=K+1
140 GOTO 110
150 LET M=M/10
160 PRINT M
170 STOP
180 END
```

I dati di input N(K) possono essere sia interi che decimali, la media M è un numero decimale.

Codifica BASIC del programma di pag. 13:

```
10 REM CALCOLO DELLA MEDIA ARITMETICA DI 10 NUMERI INTERI
20 LET M=0
30 LET K=1
40 INPUT"SCRIVI UN NUMERO INTERO",N%
50 LET M=M+N%
60 IF K=10 GOTO 90
70 LET K=K+1
80 GOTO 40
90 LET M=M/10
100 PRINT M
110 STOP
120 END
```

I dati di input sono numeri interi, la media M è un numero decimale.

3.3. Norme operative per il PET

Nella parte posteriore sinistra del calcolatore c'è un interruttore per dare corrente al PET. Quando date corrente il PET inizializza la sua memoria interna, cancella lo schermo e quindi visualizza su di esso il seguente messaggio:

***COMMODORE BASIC
xxxx BYES FREE

dove xxxx è = 31743 per PET 32 K, 7167 per PET 8k, ecc.
A questo punto il calcolatore è pronto a colloquiare con l'utente in linguaggio BASIC, sotto controllo del suo SISTEMA OPERATIVO.

Il *Sistema Operativo* è formato da un gruppo di programmi che sono permanentemente registrati nella *memoria* ROM del PET. Memoria ROM significa Memoria A Sola Lettura (Read Only Memory), cioè che non può essere scritta dall'utente. Il numero 31743 che compare nel messaggio seguito da BYTES FREE, significa che l'utente ha a disposizione 31743 BYTES (gruppi di 8 bits) di memoria RAM (Random Access Memory), cioè di memoria nella quale l'utente può scrivere. Vedrete che il *cursore* lampeggiante del video si ferma sotto la R di READY. Il calcolatore è ora in uno stato che chiamiamo *command level,* cioè livello di comando ed è pronto a ricevere i comandi consentiti dal BASIC. Vediamo alcuni di questi comandi:

Comando	Scopo del comando
NEW	cancella quello che si era messo prima in memoria e predispone l'entrata di un nuovo programma BASIC.
LIST	evidenzia sullo schermo il programma che è in memoria.
LIST x-	evidenzia sullo schermo il programma dalla linea x in avanti.

Comando	Scopo del comando
LIST x	evidenzia solo la linea x.
LIST -x	evidenzia il programma dall'inizio fino alla linea x.
LIST x-y	evidenzia il programma dalla linea x alla linea y.
RUN	inizia l'esecuzione del programma che si trova in memoria dalla linea avente numero di linea minore, dopo aver inizializzato a zero o blank tutte le variabili ed avere riinizializzato i blocchi dei DATA (che studieremo in seguito).
RUN x	fa partire il programma dalla linea x

Se volete scrivere in memoria un programma, dovete usare il comando NEW e poi scrivere il programma una linea dopo l'altra. Dopo ogni linea di programma si deve usare il tasto RETURN. Se vi accorgete di aver *dimenticato una linea di programma,* per esempio avete scritto le linee 30 e 50 dimenticando la 40, scrivete la 40, dopo la 50; pensa il BASIC a mettere le linee nel giusto ordine in base al numero di linea. Se vi accorgete di aver fatto un errore che comporta la *cancellazione di una linea di programma,* potete ottenere di cancellare la linea scrivendo il numero della linea seguito da RETURN. Comunque se avete scritto la linea 60 in modo errato e riscrivete una nuova linea 60, sostituisce la precedente. Se vi accorgete di aver *sbagliato un carattere,* subito dopo averlo battuto, usate il tasto DEL, il cursore torna indietro ed il carattere viene cancellato. Se volete *cancellare* un carattere qualunque di una riga, dovete portare il cursore, usando i tasti di movimento cursore, al carattere a destra di quello da cancellare e poi usare il tasto DEL; vedrete che il carattere viene cancellato e tutti i caratteri a destra si spostano verso sinistra di una posizione, cioè non resta il buco bianco. Naturalmente queste correzioni si devono fare prima di usare il tasto RETURN; infatti quando si usa il tasto RETURN la linea di programma va in memoria. Se volete invece *inserire un carattere* in una posizione della linea, dovete portare il cursore alla posizione seguente a quella interessata ed usare il tasto INST, tenendo premuto il tasto SHIFT: con questa operazione si crea uno spazio per l'inserimento ed il cursore si sposta alla posizione appena guadagnata, se ora premete un tasto qualunque, il carattere viene inserito al posto dello spazio. Se volete *correggere* qualcosa in una linea per la quale è già stato dato il RETURN, dovete riscriverla tutta, oppure raggiungerla con il cursore e provvedere alla correzione. La cosa più comoda è richiamare la linea da correggere con: LISTm, nella parte bassa libera dello schermo e provvede alla correzione. Questo modo di procedere è necessario, se la linea si allunga, in seguito alle correzioni, e va ad invadere le linee seguenti. Quanto avete terminato di scrivere il programma potete, usando il tasto LIST, *riottenere la lista* del programma. Se volete *far girare il programma,* dovete usare il tasto RUN.

Quando il programma è terminato, cioè il controllo è tornato al sistema, vedrete comparire la scritta READY.

Abbiamo visto che il calcolatore si può trovare in due stati:

- COMMAND LEVEL
- PROGRAM LEVEL.

Il calcolatore si trova in Program Level dopo il RUN e fino all'apparire di nuovo della scritta READY.

Fino ad ora abbiamo visto come si scrive ed adopera un programma BASIC. Dobbiamo ora occuparci del funzionamento "diretto" delle frasi BASIC. Se scriviamo le frasi BASIC senza il numero di linea, quando si usa il RETURN la frase viene subito eseguita e non solo memorizzata (come avviene se c'è il numero di linea). Naturalmente non tutte le frasi BASIC studiate possono essere usate in modo diretto; non possono essere usate in modo diretto quelle che fanno riferimento a numeri di linea. Questo modo del PET lo fa funzionare come una super calcolatrice.

Se scriviamo: PRINT 4x6 lo schema visualizza 24.

Provate ad usare il PET in modo diretto e capirete meglio la differenza tra uso "diretto" ed uso "differito", cioè sotto controllo del programma memorizzato. Comunque, se durante lo svolgimento di un programma, arrivate ad uno STOP o ad un END e desiderate scrivere ed eseguire alcune operazioni in modo diretto, lo potete fare senza danneggiare il programma memorizzato. Tenete presente che quando scrivete una frase PRINT, potete sostituire il PRINT con un ? (punto interrogativo); il sistema quando riceve il ? lo memorizza come PRINT e se listate il programma con LIST, vedrete comparire il PRINT.

3.4. La tastiera e lo schermo del PET

Le figure 3-1a e 3-1b riportano le tastiere del 2001 e del 3032.
Nella figura 3-1b si può notare che i tasti hanno una conformazione tale per cui si vede un carattere sovraimpresso e uno nella parte frontale.

Nella figura 3-1a, invece, su ogni tasto sono riportati due simboli. Potete notare che la tastiera è divisa in due parti: verso destra ci sono le cifre numeriche. Il carattere riportato sopra ogni tasto (figura 3-1b) è quello che si usa in condizioni normali.

(concetto di scrittura minuscola con la normale macchina da scrivere), mentre quello riportato frontalmente si usa tenendo schiacciato il tasto SHIFT. Il tasto SHIFT è presente due volte a destra e a sinistra. Il tasto SHIFT LOCK serve per fissare il tasto SHIFT. Il tasto lungo in basso, sul quale è stato scritto SPAZIO, serve per spaziare; sul calcolatore è bianco senza scritta.

Nella tastiera sono presenti tutti i normali caratteri delle macchine da scrivere più altri caratteri aggiuntivi, che si usano frequentemente nelle apparecchiature di input dei calcolatori. Inoltre sulle tastiere viste sono disponibili i caratteri grafici (parte bassa dei tasti, resi attivi da SHIFT) che permettono di disegnare sullo schermo, sia in modo diretto che sotto controllo del programma.

Lo schermo del PET può contenere 25 righe di 40 caratteri, ciascuna; quando lo schermo è pieno, cioè contiene 1000 caratteri si ha lo scorrimento automatico verso l'alto. All'interno del calcolatore esiste una memoria per lo schermo, TV RAM, di 1 K (1024 bytes) che può contenere tutto lo schermo. Esiste un programma che chiamiamo *programma di correzione*

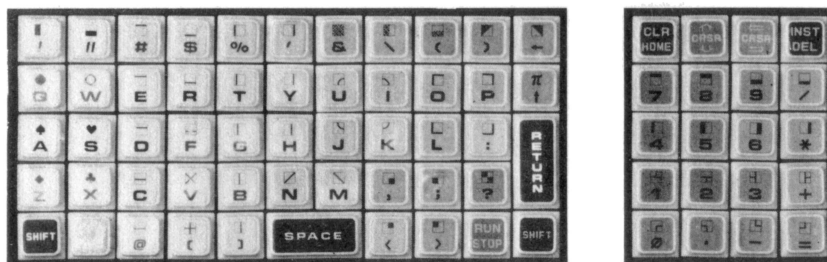



Figura 3-1a. Esempio di tastiera 2001.

Figura 3-1b. Esempio di tastiera 3032.

Figura 3-1c. Esempio di tastiera CBM 8000.

dello schermo, che entra in azione quando si dà corrente al calcolatore e compare la scritta:

> ***COMMODORE BASIC***
> 31743 BYTES FREE
> READY

con il cursore lampeggiante sotto la R di READY.

A questo punto la tastiera è abilitata e qualunque cosa si scrive sulla tastiera, questo compare sullo schermo. Tenete presente che il PET si trova in uno stato nel quale attende o comandi o istruzioni BASIC, per cui se voi scrivete sulla tastiera per esempio:

SONO CONTENTO DI IMPARARE A USARE IL PET
seguito dal tasto RETURN

vedrete comparire:

SONO CONTENTO DI IMPARARE A USARE IL PET
?SYNTAX ERROR
READY

Infatti la frase che avete scritto non è né un comando né una istruzione BASIC diretta e quindi il sistema non la riconosce e vi segnala che c'è un errore di sintassi.

Comunque non preoccupatevi di questo SINTAX ERROR e fate molte prove per imparare ad usare i tasti di comando della tastiera che agiscono sullo schermo. Ricordate che quello che scrivete sullo schermo va in memoria solo se usate il tasto RETURN. Se provate a scrivere una linea di più di 40 caratteri, vedrete che il PET va a capo da solo e continua. In condizioni normali (senza tasto SHIFT premuto) il PET scrive in stampatello, se volete i caratteri corsivi dovete operare così: POKE 59468,14 RETURN con

questa istruzione diretta memorizzate nel byte 59468 il numero 14. In questa situazione il PET, invece di usare il primo insieme di caratteri di cui dispone normalmente, usa il secondo insieme di caratteri. La differenza tra i primo ed il secondo insieme di caratteri è la seguente: nel secondo insieme in condizioni normali si hanno i caratteri minuscoli corsivi. Per ottenere le maiuscole si deve adoperare lo SHIFT.

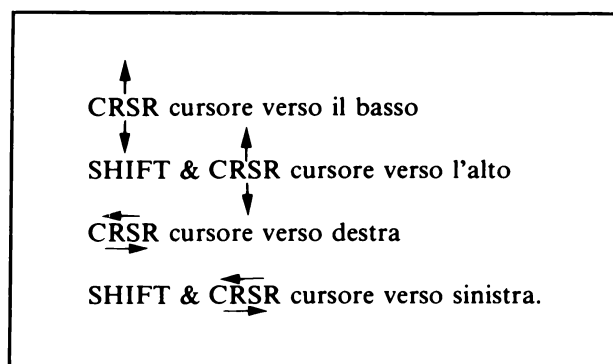
Se volete sapere quale valore è contenuto nel byte 59468 dovete operare così:
?PEEK (59468) RETURN

vedrete comparire:

14 se agisce il secondo set di caratteri,

12 se agisce il primo set di caratteri.

Se volete tornare al primo set di caratteri, dovete operare così:
POKE 59468,12 RETURN

Vediamo come si *usano i tasti di comando della tastiera e quali effetti* hanno sullo schermo. Premendo il tasto HOME si ha lo spostamento del cursore lampeggiante nell'angolo in alto a sinistra dello schermo. Premendo il tasto SHIFT & CLEAR (& significa insieme, cioè contemporaneamente) si ottiene la cancellazione dello schermo ed il posizionamento del cursore in alto a sinistra. Per spostare il cursore si hanno a disposizione i due tasti CRSR rispettivamente con freccia alto/basso e sinistra/destra. Provate ad usarli e vedrete che se, usate il primo, il cursore si sposta verso il basso, mentre se usate il secondo il cursore si sposta verso destra. Se volete andare o verso l'alto o verso sinistra, dovete usare i due tasti tenendo schiacciato il tasto SHIFT. Quindi si ha:

Per fare cancellazioni ed inserzioni su una linea dello schermo, si ha a disposizione il tasto con il doppio uso: DEL o SHIFT & INST.

Dell'uso di questo tasto abbiamo già parlato a pagina 26, riepiloghiamo comunque le regole:

- DEL cancella il carattere che precede il cursore e sposta tutti i caratteri restanti della linea, se ce ne sono, di una posizione a sinistra;
- SHIFT & INST. crea uno spazio tra la posizione del cursore ed il carattere seguente, il cursore si porta nella posizione dello spazio. Premendo un qualunque tasto si inserisce il nuovo carattere.

Il PET scrive normalmente sullo schermo con caratteri verdi in campo nero. Il tasto OFF/RVS serve per ottenere il carattere nero in campo verde. Se si preme il tasto RVS si ha il cambio a fondo verde e carattere nero fino o al RETURN o all'uso del tasto SHIFT & OFF. Il tasto OFF/RVS può essere usato anche in fase di stesura programmi all'interno di stringhe alfanumeriche come normale carattere di controllo per ottenere scritte particolarmente evidenziate. Durante la lista di un programma lungo il contenuto sullo schermo si muove verso l'alto per eliminare le prime righe ed accettare le nuove, con movimento abbastanza veloce. Se si vuole rallentarlo, premere il tasto OFF/RVS; se si vuole fermarlo, premere STOP.

A questo punto è consigliabile esercitarsi alla tastiera per provare tutti i tasti di comando e provare i 3 programmi esempio del paragrafo 3.2.

3.5. Alcuni esercizi di programmazione in BASIC

Vi proponiamo alcuni esercizi per consolidare le nozioni apprese prima di procedere nello studio del PET e del linguaggio BASIC.

Esercizio 1: Scrivere un programma che:

1. legga dalla tastiera 15 numeri decimali e li memorizzi in un vettore NN;
2. metta in ordine crescente i 15 numeri usando sempre il vettore NN;
3. stampi il vettore NN.

Analisi del problema: per il punto 1. si deve istituire un ciclo di lettura che legga i 15 numeri; i numeri possono essere letti uno alla volta o a gruppi; decidiamo di leggerli a gruppi di 3 per volta. Nel programma avremo quindi 5 letture di 3 numeri per volta. Per poter memorizzare i numeri, all'inizio del programma useremo la frase DIM NN(14), con questa frase si dimensiona un vettore di 15 posizioni, per il qualle l'indice varia da 0 a 14. NN (senza %) dice che i numeri sono decimali. I numeri di NN possono anche essere negativi. Per il punto 2. si deve trovare un algoritmo di ordinamento che possa essere facilmente programmato e che ordini i numeri nello stesso vettore che li contiene inizialmente. Possiamo agire in questo modo:

- confrontiamo il primo numero di NN con tutti gli altri, uno per volta. Se troviamo un numero minore del primo, operiamo un doppio

scambio tra i due numeri, in modo che nella prima posizione di NN venga sempre a trovarsi un numero minore degli altri. Quando abbiamo fatto l'ultimo confronto, cioè del primo numero con il quindicesimo (NN(0) con NN(14)), siamo sicuri che nella prima posizione di NN si trova il minore dei 15 numeri;

- ripetiamo quanto sopra partendo dal secondo numero di NN; in questo caso faremo in tutto 13 confronti ed avremo nella seconda posizione di NN il numero immediatamente superiore a quello che sta nella prima posizione;
- ripetiamo questo ciclo in tutto 14 volte, infatti nell'ultimo ciclo confronteremo NN(13) con NN(14) ed avremo il vettore NN in ordine crescente;

per ottenere quanto detto sopra, ci serviremo di due *puntatori* o indici: K per puntare alla prima posizione del confronto ed I per puntare alla seconda posizione del confronto.

Per il punto 3. si deve istituire un ciclo di visualizzazione dei 15 numeri; si adopera il punto e virgola per non andare a capo. Alla fine si da un PRINT a vuoto per andare a capo come si vede nel diagramma a blocchi che descrive graficamente questo algoritmo.

Diamo, inoltre, in forma discorsiva l'algoritmo risolutivo del problema, per rendere più semplici le cose a coloro che si accostano per la prima volta alla programmazione:

1. 1a inizio del ciclo di lettura dei numeri NN, a 3 a 3;
   poniamo K = 0;
   2a chiediamo 3 numeri alla tastiera e li memorizziamo in NN(K), NN(K + 1) e NN(K + 2);
   3a confrontiamo K con 12, infatti K = 12 quando leggiamo gli ultimi 3 numeri che vanno rispettivamente in NN(12), NN(13) e NN(14). Se K = 12 il ciclo delle 5 letture è terminato, mentre se K è diverso da 12, si deve proseguire dal punto 4a. Se il ciclo è terminato si salta al punto 1b);
   4a incrementiamo K di 3 e torniamo al punto 2a;

2. 1b inizia il ciclo di ordinamento. Poniamo K = 0 per puntare al primo elemento del vettore NN;
   2b poniamo I = K + 1 per puntare al secondo elemento del confronto;
   3b confrontiamo NN(K) con NN(I); se NN(K) è minore o uguale a NN(I), si prosegue dal punto 5b, se no si prosegue dal punto 4b;
   4b si scambiano tra loro i contenuti di NN(K) e di NN(I) servendosi di una variabile di comodo ausiliaria C e si prosegue dal punto 5b. Il numero minore è andato nella posizione del primo termine del confronto;

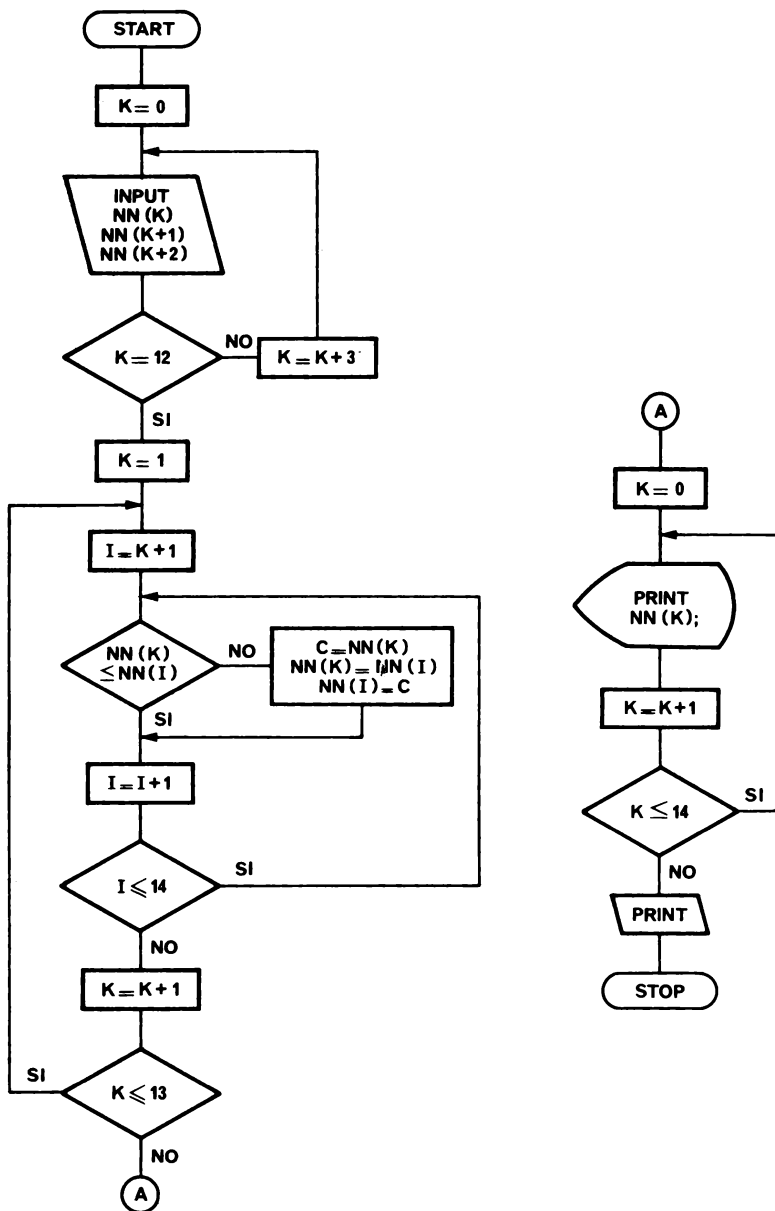



Figura 3-2. Ordinamento di una tabella di numeri.

5b incrementiamo I di 1 per considerare un nuovo numero di NN come secondo termine del confronto;

6b confrontiamo I con 14; se I è minore o uguale a 14, il ciclo di confronti non è terminato e si prosegue dal punto 3b, se I è maggiore di 14 il ciclo di confronti è terminato e si deve cambiare il primo termine del confronto, sempre che già non sia terminato l'ordinamento;

7b incrementiamo K di 1 per puntare al numero, seguente l'ultimo sistemato, come primo termine del confronto;

8b confrontiamo K con 13; se K minore o uguale a 13, si torna al punto 2b per iniziare un nuovo ciclo di confronti. Se K maggiore di 13 si prosegue dal punto 1c perchè l'ordinamento è terminato. Osserviamo che in ogni ciclo di confronti, il numero dei confronti da effettuare diminuisce di uno;

3. 1c inizia il ciclo di stampa. Poniamo K = 0 per puntare al primo elemento del vettore;

   2c evidenziamo NN(K) senza andare a capo;

   3c incrementiamo K di 1 per passare all'elemento seguente;

   4c confrontiamo K con 14; se K è minore o uguale a 14, torniamo al punto 2c, se no proseguiamo;

   5c diamo un comando di PRINT a vuoto per andare a capo e l'algoritmo è terminato.

Codifica BASIC del problema proposto: Programma ORD

```
5 REM PROGRAMMA ORD
10 REM ORDINAMENTO DI UN VETTORE DI 15 NUMERI DECIMALI
20 REM FASE DI LETTURA DEI NUMERI 3 A 3
25 DIM NN(14)
30 LET K=0
35 PRINT "SCRIVI 15 NUMERI 3 A 3 "
40 INPUT NN(K),NN(K+1),NN(K+2)
50 IF K=12 THEN 70
60 LET K=K+3
65 GO TO 40
70 REM FASE DI ORDINAMENTO DEL VETTORE NN
80 LET K=0
90 LET I=K+1
100 IF NN(K)<=NN(I) GOTO 140
110 LET C=NN(K)
120 LET NN(K)=NN(I)
130 LET NN(I)=C
140 LET I=I+1
150 IF I<=14 GOTO 100
160 LET K=K+1
170 IF K<=13 THEN 90
180 REM FASE DI STAMPA DEL VETTORE NN ORDINATO
190 LET K=0
200 PRINT NN(K)
210 LET K=K+1
220 IF K<=14 GOTO 200
230 PRINT
240 STOP
```

Facciamo notare che l'istruzione 30 LET K = 0, poteva anche essere omessa, dato che quando si dà il RUN del programma le variabili vengono azzerate dal sistema. Abbiamo preferito scrivere tale istruzione per evidenziare che all'inizio di un ciclo il contatore di ciclo deve essere inizializzato.

Esercizio 2: Vi proponiamo di apportare delle modifiche all'esercizio 1, per esempio, modificare il ciclo di lettura dei numeri leggendone 5 per volta, oppure modificare il ciclo di stampa usando la virgola invece del punto e virgola. Potete provare a modificare il ciclo di stampa in modo da stampare 5 numeri per riga. Potete cercare un diverso algoritmo di ordinamento. Potete fare l'analisi, il diagramma e la codifica usando un algoritmo che metta in ordine i numeri nel ciclo di lettura, cioè: leggete un numero per volta e prima di sistemarlo nel vettore NN cercate di stabilire quale posizione gli compete in modo che il vettore risulti già ordinato.

Esercizio 3: Scrivere un programma che:

1. chieda alla tastiera un numero compreso tra 1 e 365, naturalmente intero,
2. calcoli a quale giorno della settimana quel numero corrispondeva nell'anno 1979, tenendo presente che il giorno 1/1/79 era lunedì.

Vi ricordiamo che dividendo un numero per 7, i possibili resti sono 0,1,2,3,4,5,6. Il resto 0 corrisponde alla domenica, mentre il resto 1 corrisponde al lunedì, ecc. Lasciamo a voi il compito di stendere una breve analisi, il diagramma a blocchi e la codifica di questo programma.

3.6. Altre frasi BASIC

In questo paragrafo ci occupiamo delle istruzioni:

READ, DATA, RESTORE

Le istruzioni READ e DATA servono per poter memorizzare dei dati all'interno del programma, in blocchi di dati consecutivi, e richiamarli quando servono. Facciamo un esempio:

- in un programma abbiamo bisogno di esaminare in sequenza, cioè uno dopo l'altro, una serie di 10 numeri interi;
- scriviamo questi 10 numeri in una frase DATA così:

  50 DATA 4,27,87,342,675,456,78,98,110,354;
- il sistema memorizza questi 10 numeri a partire da una determinata posizione di memoria e predispone un suo puntatore interno al primo di questi 10 numeri, puntatore di cui il programmatore non si accorge;
- nel programma per avere disponibili questi numeri in sequenza partendo dal primo, scriviamo: 100 READ N; in N abbiamo la prima volta che viene eseguita la frase READ, il primo numero del blocco, e cioè 4;

- se eseguiamo un READ successivo al primo, avremo disponibile il secondo numero, e cioè 27 e così via;
- cioè ogni volta che si ha il READ di una qualunque variabile, viene trasferito nella variabile il numero al quale si trova il puntatore interno ed il puntatore avanza al prossimo numero; se ad un certo punto i numeri del blocco sono esauriti, si ha il messaggio di errore ?OUT OF DATA;
- se si desidera riposizionare il puntatore interno al primo numero del blocco, si deve usare la frase: RESTORE, al successivo READ viene reso disponibile il primo numero del blocco.

In un programma si possono avere diverse frasi DATA. Il sistema prepara un unico blocco di dati prendendo i diversi DATA nell'ordine dei numeri di linea a cui appartengono e gestisce i blocco dei dati con un unico puntatore interno. Il formato della frase è:

DATA lista di dati separati da virgole

Consideriamo questo esempio:

```
10 DATA 10,20,30,40,50,60,70
20 I = 1
30 READ A
40 PRINT A;
50 I = I + 1
60 IF I < 8 THEN 30
70 RESTORE
80 GOTO 20
```

se lo provate, vedrete che stamperà continuamente così:

10 20 30 40 50 60 70 10 20 30 40 50 60 ecc.

andando automaticamente a capo quando ha raggiunto i 40 caratteri per riga dello schermo. Per fermarlo, dovrete premere il tasto STOP. È buona norma in un programma scrivere tutte le frasi DATA in frasi consecutive per avere una visione d'insieme del blocco di dati interno al programma. L'uso del DATA/READ/RESTORE può essee utile in un programma per memorizzare tabelle che non variano molto spesso. Sarebbe più lungo memorizzare tali dati con una serie di istruzioni LET.

Esercizio 4: Scrivere un programma che:

1. chieda alla tastiera una data nella forma GG,MM,AA dove:
   - GG = giorno,
   - MM = mese in numero da 1 a 12
   - AA = anno solo le ultime due cifre;
2. stampi al video la data nella forma GG mese in lettere 19AA.

Analisi: l'algoritmo è molto semplice. Si devono memorizzare all'interno del programma le descrizioni dei mesi ed andare a prendere quella giusta in

base al numero del mese. Si deve poi aggiungere 1900 alle due ultime cifre dell'anno. Per memorizzare le descrizioni dei mesi, usiamo la frase DATA. Per puntare al mese giusto, inizializziamo un contatore K al valore 1 e diamo dei READ in ciclo fino a quando K = MM, incrementando K di 1 ogni volta. Per aggiustare l'anno basta sommargli 1900. Se si da il mese errato, cioè non compreso tra 1 e 12, si avrà errore perchè si cercherà di fare più di 12 READ. Tracciamo il diagramma a blocchi e lo facciamo seguire dalla codifica del programma in BASIC.

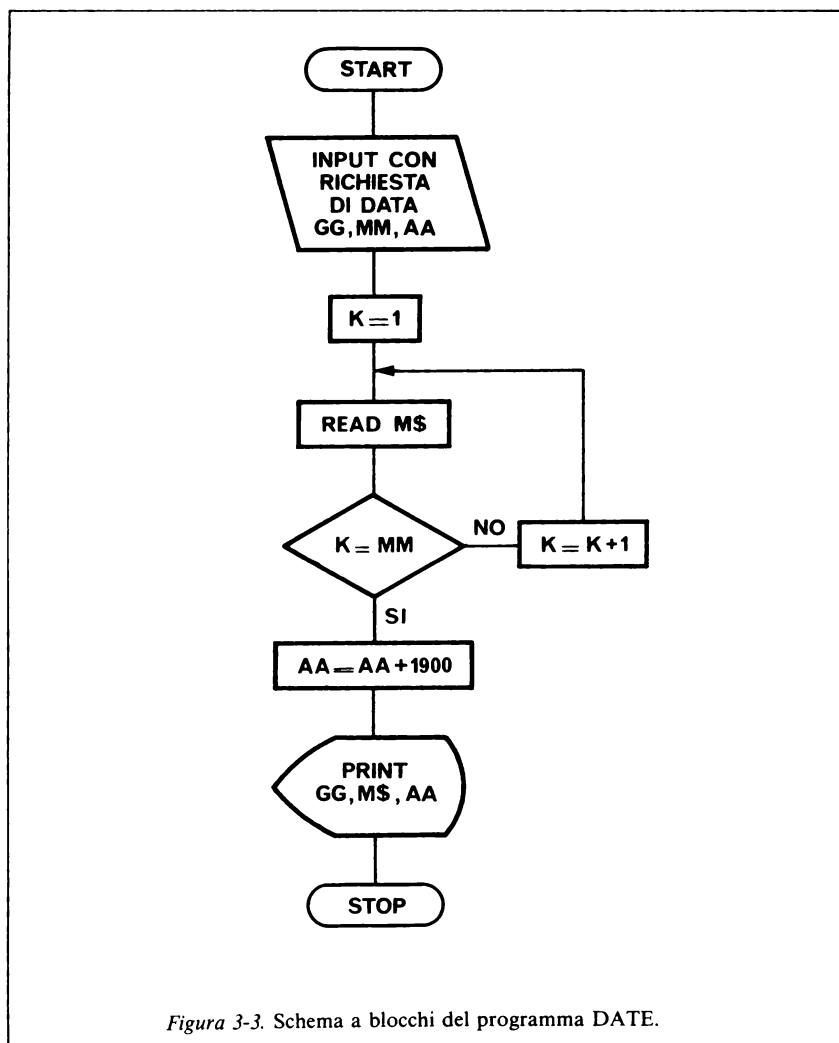



Figura 3-3. Schema a blocchi del programma DATE.

Programma DATE

```
10 REM PROGRAMMA STAMPA DATA CON MESE IN LETTERE
20 DATA GENNAIO,FEBBRAIO,MARZO,APRILE
30 DATA MAGGIO,GIUGNO,LUGLIO,AGOSTO
40 DATA SETTEMBRE,OTTOBRE,NOVEMBRE,DICEMBRE
50 INPUT"SCRIVI DATA:GG,MM,AA";GG,MM,AA
60 K=1
70 READ M$
80 IF K =MM THEN 110
90 K=K+1
100 GO TO 70
110 AA=AA+1900
120 PRINT GG;M$;AA
130 STOP
```

Si sono usate le variabili GG,MM,AA che sono decimali, ma si possono usare anche per numeri interi.

Esercizio 5: Scrivere un programma che:

1. legga dalla tastiera un numero intero N%;
2. ricerchi in una tabella di 50 numeri, che è interna al programma, se esistono numeri uguali a quello letto N% ed evidenzi il risultato al video. La tabella di numeri non è ordinata e quindi la ricerca va fatta su tutta la tabella.

Analisi: l'algoritmo è molto semplice, si tratta di istituire un ciclo che renda disponibile con READ uno alla volta i numeri che sono memorizzati internamente al programma e faccia il confronto con N%. Se trova uguaglianza, invii un messaggio di segnalazione al video. Il ciclo va percorso 50 volte. Si possono contare le uguaglianze riscontrate. Segue il diagramma a blocchi (figura 3-4) e la codifica in BASIC.

Programma RICER

```
10 REM RICERCA PER UGUALE IN UNA TABELLA
20 REM DISORDINATA DI NUMERI
30 DATA24,67,89,65,34,9,7,9,3,789
40 DATA67,34,56,789,999,444,456,34,5,7
50 DATA7,5,1,1,89,765,30234,18,99,10
60 DATA1,2,3,4,5,6,7,8,9,10
70 DATA21,22,23,24,25,29,30,5,24,99
80 INPUT"SCRIVI UN NUMERO INTERO";N%
90 C%=1
100 K%=1
110 READ M%
120 IF M%=N% THEN 140
130 GO TO 160
140 PRINT"SI E' TROVATO UGUALE";C%;"VOLTE"
150 C%=C%+1
160 IF K%=50 THEN 190
170 K%=K%+1
180 GOTO 110
190 STOP
```

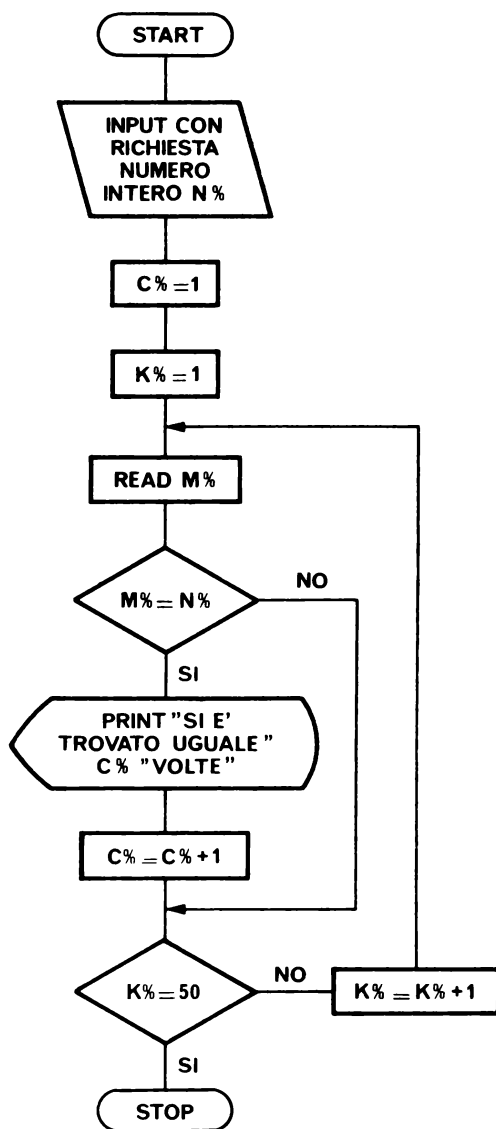



Figura 3-4. Schema a blocchi del programma RICER.

Esercizio 6: Si modifichi il programma precedente aggiungendo un controllo finale per C%; se C% è rimasto = 1, si stampi un messaggio che dica di non aver mai trovato uguaglianza.

Esercizio 7: Si scriva un programma che:

1. legga dalla tastiera un numero decimale N;
2. ricerchi in una tabella di 50 numeri decimali che è interna al programma, ma è ordinata già in ordine crescente, se trova una o più uguaglianze (naturalmente consecutive) e stampi un messaggio di risposta sia in caso positivo che in caso negativo;
3. stampi i 50 numeri 5 numeri per riga.

Lasciamo al lettore il compito di risolvere questo problema.

3.7. Norme operative per l'uso della cassetta C2N

Fino ad ora ci siamo riferiti al PET fornito solo di tastiera e video. Vediamo ora come si usa il nastro a cassetta direttamente collegato al PET. Questa unità per il PET è una *periferica*. Le periferiche sono riconosciute dal sistema mediante un numero logico. Il numero attribuito alla cassetta magnetica è 1, se la cassetta è una sola. Sull'unità ci sono 5 tasti di comandi, non c'è interruttore di accensione, infatti la cassetta è attivata quando il PET riceve corrente.

I tasti sono:

- REC (RECORD),
- REW (REWIND),
- F.FWD (FAST FWD),
- PLAY,
- STOP/EJECT.

Il tasto STOP serve per aprire l'alloggiamento della cassetta e per annullare gli altri tasti. Il tasto REW serve per riavvolgere il nastro. Il tasto F.FWD serve per fare avanzare velocemente il nastro. Il tasto PLAY verrà spiegato tra poco. Per il momento ci occupiamo della procedura per registrare sulla cassetta programmi che si trovano in memoria, per verificare se sono stati registrati bene e per caricare dalla cassetta in memoria dei programmi.

Quando vogliamo usare una cassetta, dobbiamo inserirla nel suo contenitore, e se vogliamo posizionarla all'inizio, premere il tasto REW ed attendere che si fermi, allora premere STOP. Supponiamo di avere in memoria un programma, di averlo già provato e di volerlo conservare sulla cassetta. Per poter fare ciò, dobbiamo attribuire un nome al programma; con tale nome il programma verrà riconosciuto quando lo vorremo ricercare sulla cassetta e mediante tale nome si distingue dagli altri programmi eventualmente memorizzati sulla stessa cassetta. Nelle pagine precedenti è stato attribuito un nome ad ogni programma svolto. Se abbiamo in memo-

ria il programma DATE, che stampa una data con il mese in lettere, dobbiamo dare questo comando:

SAVE "DATE" RETURN

il sistema risponde:

PRESS PLAY & RECORD,

ed allora l'operatore deve *premere contemporaneamente* (& significa contemporaneamente) i due tasti PLAY e REC. La cassetta registra e sul video appare:

OK
WRITING DATE   e quando ha finito di registrare:
READY

A questo punto sulla cassetta è stato registrato il programma DATE. Se volete *verificare la bontà della registrazione* potete operare così: fate riavvolgere la cassetta con il tasto REW e poi STOP. In memoria avete sempre il vostro programma appena registrato. Ricordate che il programma si cancella dalla memoria solo se battete NEW alla tastiera. Scrivete: VERIFY "DATE" RETURN il sistema risponde: PRESS PLAY e voi premete il tasto PLAY appare sul video: FOUND DATE ed alla fine della verifica READY. Se la verifica non va bene appare: VERIFY ERROR e dovete rifare la registrazione. Ora dovete provare a *caricare il programma dalla cassetta* in memoria. Operate così:

- battete NEW RETURN per cancellare la memoria → appare READY;
- premete REW e poi STOP alla cassetta;
- battete LOAD "DATE" RETURN appare PRESS PLAY
- premete PLAY appare OK
  SEARCHING FOR DATE
  FOUND DATE
  LOADING
  READY
- battete LIST appare la lista del vostro programma.

Riepiloghiamo i comandi studiati:

- SAVE "nome" RETURN per memorizzare un programma su cassetta;
- LOAD "nome" RETURN per caricare da cassetta in memoria un programma
- VERIFY "nome" RETURN per verificare la bontà di una registrazione.

Figura 3-5. Registratore a cassetta del PET Commodore.

Sulla cassetta possono essere registrati successivamente diversi programmi, ma bisogna fare attenzione a non cancellare quello che c'è già, quando si vuole aggiungere qualcosa di nuovo. All'esterno della cassetta bisogna annotare ordinatamente i nomi dei programmi registrati. Supponiamo di avere già registrato su una cassetta il programma APROG1 e di voler registrare un nuovo programma di nome BPROG2 subito dopo il precedente. Il programma BPROG2 è già in memoria. Operiamo così:

- montiamo la cassetta e con REW la riavvolgiamo;
- battiamo: VERIFY "APROG1";
- risponde PRESS PLAY lo facciamo;
- risponde FOUND APROG1 e comincia a verificare APROG1 con il nostro BPROG2 che è in memoria; alla fine della verifica ovviamene dice VERIFY ERROR, infatti ha confrontato due programmi diversi. Però il nastro si è svolto ed ha superato tutto APROG1;
- a questo punto *senza dare REW* diamo il comando SAVE "BPROG2" RETURN e BPROG2 va a registrarsi in coda ad APROG1.

Un altro modo per fare registrazioni successive è quello di scrivere un programma e di comandarne il SAVE, poi battere NEW, scrivere in memoria un secondo programma e comandarne il SAVE e così via senza mai riavvolgere il nastro tra un programma e l'altro. Quando un nastro contiene più di un programma e si usa il comando LOAD, il sistema fa scorrere la cassetta e ricerca il programma voluto.

CAPITOLO 4

LE FRASI NECESSARIE PER COMPLETARE LO STUDIO DEL LINGUAGGIO BASIC

4.1. Le istruzioni per il controllo dei cicli

Nel capitolo precedente abbiamo visto che in programmazione capita di frequente di dover controllare lo svolgimento di cicli ripetitivi. Abbiamo visto come si usa un contatore ricorrendo a:

- inizializzazione,
- incremento,
- controllo.

Esistono due frasi BASIC, da usare accoppiate, che permettono di controllare i cicli di programma; esse sono:

```
FOR variabile = n1 TO n2 STEP n3
...... gruppo di istruzioni che
...... devono essere ripetute
NEXT variabile
```

in cui: variabile è il contatore del ciclo, n1 è il valore iniziale che si vuole dare al contatore, n2 è il valore finale che si vuole raggiunga il contatore, n3 è l'incremento che si vuole dare ad ogni giro;
n1, n2 ed n3 possono essere sia variabili che costanti che espressioni aritmetiche valide. Si possono usare anche variabili con un solo indice. Non si possono usare variabili con il suffisso %, cioè variabili intere, o variabili con due indici. Se n3 è uguale ad 1, STEP n3 può essere omesso. Se n3 è positivo, dovrà essere $n1 <= n2$; se n3 è negativo, dovrà essere $n1 >= n2$.

Le istruzioni comprese tra il FOR e NEXT vengono eseguite almeno una volta, cioè per esempio FOR V = 1 TO 0 .... NEXT esegue le istruzioni del ciclo 1 volta. Nella frase NEXT si può omettere di citare la variabile; allora NEXT chiude l'ultimo FOR incontrato nel programma. Le istruzioni FOR-/NEXT devono essere sempre bilanciate altrimenti si ha errore. Esempio:

stampa dei primi 100 numeri interi

```
10 REM STAMPA DEI PRIMI 100 NUMERI INTERI
20 FOR I = 1 TO 100
30 PRINT I;
40 NEXT I
50 PRINT
```

Il programma esempio stampa i primi 100 numeri interi senza andare a capo tra un numero e l'altro (;) e dopo il 100, va a capo con il PRINT a vuoto.

Senza fare uso del FOR si sarebbe potuto programmare così:

```
10 REM STAMPA DEI PRIMI 100 NUMERI INTERI
20 I = 1
30 PRINT I;
40 IF I = 100 THEN 70
50 I = I + 1
60 GO TO 30
70 PRINT
```

Come potete vedere, l'uso del FOR .... NEXT semplifica la programmazione.

Per stampare 1 primi 100 numeri interi in ordine inverso, cioè da 100 a 1 avremmo dovuto scrivere:

```
20 FOR I = 100 TO 1 STEP —1
30 PRINT I;
40 NEXT I
50 PRINT
```

Negli esempi precedenti non si è tracciato il diagramma a blocchi perchè erano molto semplici; comunque questo non si dovrebbe mai fare e qualunque problema deve, prima di essere codificato, essere studiato arrivando anche alla stesura del diagramma a blocchi.

4.2. I cicli concatenati

Nell'Esercizio 1 abbiamo risolto un problema per il quale era necessario svolgere due cicli, uno interno all'altro, il primo controllato dalla variabile K che lavorava nell'intervallo 0 —13 ed il secondo controllato dalla variabile I che lavorava tra K + 1 e 14. Abbiamo quindi già visto un esempio di *cicli concatenati.* Riportiamo ora la codifica dello stesso esercizio facendo uso delle istruzioni FOR ... NEXT usate in modo concatenato. Con riferimento alla codifica del "programma ORD" riportato nell'Esercizio 1, scriviamo ora una nuova versione.

Programma ORD

```
10 REM ORDINAMENTO DI UN VETTORE DI 15 NUMERI DECIMALI
20 REM FASE LETTURA NUMERI 3 A 3
25 DIM NN(14)
30 PRINT"SCRIVI 15 NUMERI 3 A 3"
40 FOR K=0 TO 12 STEP 3
50 INPUT NN(K),NN(K+1),NN(K+2)
60 NEXT K
70 REM FASE ORDINAMENTO VETTORE NN
80 FOR K=0 TO 13
90 FOR I=K+1 TO 14
100 IF NN(K)<=NN(I) GOTO 140
110 C=NN(K)
120 NN(K)=NN(I)
130 NN(I)=C
140 NEXT I
150 NEXT K
180 REM FASE STAMPA VETTORE ORDINATO
190 FOR K =0 TO 14
200 PRINT NN(K);
210 NEXT K
220 PRINT
230 STOP
```

Con il ciclo FOR/NEXT da 40 a 60, vengono letti i 15 numeri. Con i due cicli concatenati da 80 a 150 viene ordinato il vettore NN. Con il ciclo da 190 a 210 viene stampato il vettore ordinato.

Un ciclo inizia quando si incontra il FOR e termina quando si incontra il primo NEXT (se tale NEXT è senza variabile, va sempre bene, ma se tale NEXT reca una variabile diversa da quella del FOR si ha errore). Se dopo un FOR se ne incontra un altro, viene aperto un nuovo ciclo, subordinato al precedente, per cui il primo NEXT che si incontra chiude il ciclo iniziato con il secondo FOR, il NEXT successivo chiude il primo FOR. Alle frasi 140 e 150 si potevano sostituire le due frasi 140 NEXT e 150 NEXT, cioè NEXT senza variabile, ma si poteva anche sostituire l'unica frase 140 NEXT I,K.

4.3. Uscita forzata da un ciclo

Può essere necessario abbandonare un ciclo di FOR/NEXT prima di aver raggiunto il valore finale della variabile che controlla il ciclo; questo può accadere se all'interno del ciclo si ha, per esempio una frase del tipo IF. È buona norma, in tali casi, forzare la variabile di controllo al valore finale e saltare al NEXT per permettere di chiudere correttamente il ciclo iniziato. Facciamo un esempio:

Esercizio 8: Si scriva un programma che operi nel seguente modo:

1. legga dalla tastiera 15 nomi e li memorizzi in un vettore di stringhe A$(15);
2. legga dalla tastiera un nome N$ e ricerchi se nel vettore A$ si trova lo stesso nome;

3. stampi il nome trovato e la posizione del vettore nella quale si trova il nome.

Saltiamo l'analisi del problema, dato che il testo è sufficientemente esplicativo e passiamo a tracciare il diagramma a blocchi.

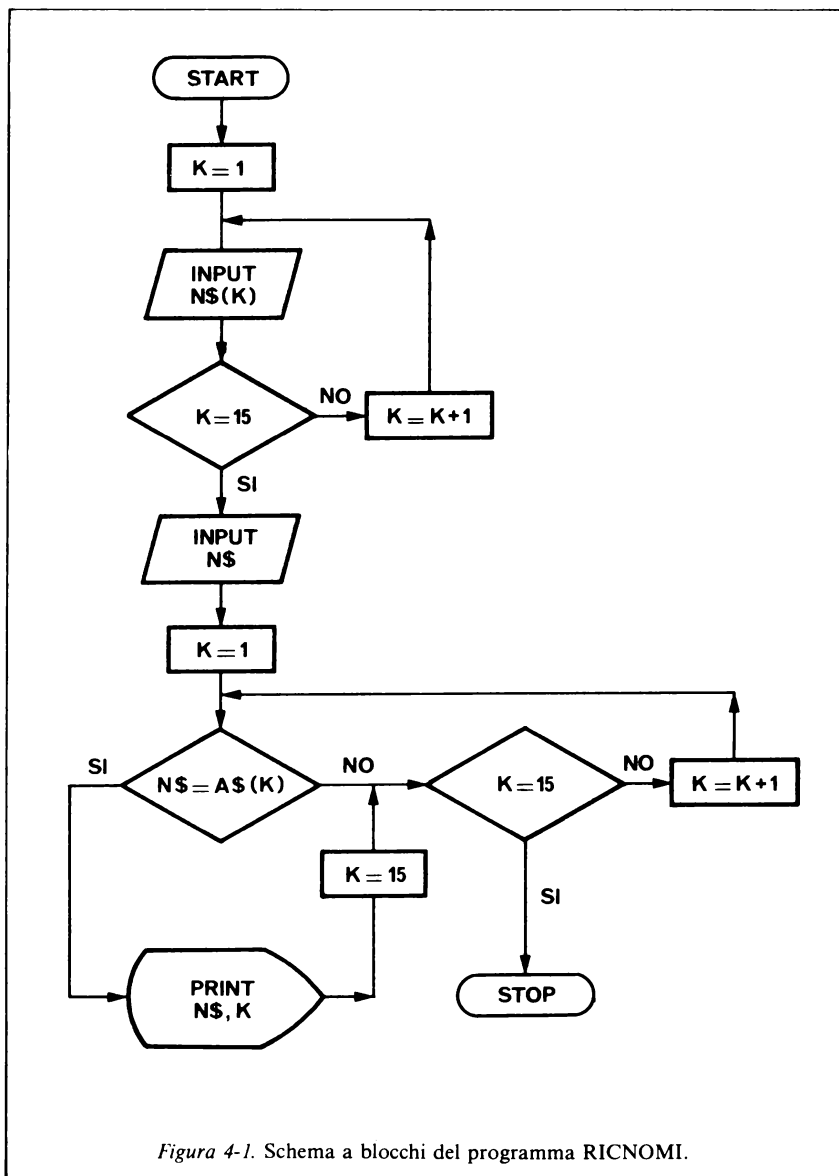



Figura 4-1. Schema a blocchi del programma RICNOMI.

Programma RICNOMI

```
10 REM RICERCA PER UGUALE IN UN VETTORE DI NOMI
20 REM FASE LETTURA 15 NOMI E LORO MEMORIZZAZIONE
30 DIM A$(15)
40 FOR K=1 TO 15
50 INPUT"SCRIVI NOME",A$(K)
60 NEXT K
70 INPUT"SCRIVI NOME PER RICERCA",N$
80 REM FASE DI RICERCA
90 FOR K=1 TO 15
100 IF N$ =A$(K) GOTO 150
110 NEXT K
120 STOP
150 PRINT"POSIZIONE  ",K,"NOME ",N$
160 K=15
170 GO TO 110
```

Nella frase 150 si evidenzia l'uguaglianza trovata, poi si pone K = 15, cioè il valore finale del controllo del ciclo e si torna al NEXT. Questo è il modo corretto di programmare una uscita anticipata da un ciclo.

Si tenga comunque presente che all'inizio del ciclo FOR vengono memorizzati i valori attuali di n1, n2 ed n3 in delle variabili di comodo, per cui anche se si variano n1, n2, n3 durante il ciclo, il numero di interazioni dipende solo dai valori iniziali. Esempio:

```
10    FOR I = X1 TO X2 STEP X3
      .
      .
      .
      .
40    X1 = X1 + 1
50    X2 = X2/3
      .
      .
      .
      .
100   NEXT I
```

Se inizialmente X1 = 1, X2 = 10, X3 = 1, tale ciclo è percorso 10 volte, anche se all'uscita X1 = 1 e X2 = 10

4.4. Salto calcolato usando la frase ON ... GOTO

La frase ON variabile GOTO n1,n2,n3, ... permette di fare un salto calcolato in base al valore della variabile. Facciamo un esempio:

```
150 ON I GOTO 200,300,400
160 PRINT"ERRORE DI GOTO CALCOLATO"
```

con la frase 150 si ottiene che:

- se I = 1 il programma prosegue dalla linea 200,

- se I = 2 il programma prosegue dalla linea 300,
- se I = 3 il programma prosegue dalla linea 400,
- se I = 4 o un numero diverso da 1,2,0,255 il programma stampa ERRORE DI GO TO CALCOLATO,
- se I = 0 o 255 il programma da l'errore di sistema per il ON..GOTO.

La frase 150 si poteva sostituire con le frasi seguenti:

```
150 IF I = 1 THEN 200
151 IF I = 2 THEN 300
152 IF I = 3 THEN 400
```

È evidente la comodità di questa frase. Si possono usare tanti numeri di linea, quanti sono contenuti in una linea di programma.

4.5. Le frasi POKE e PEEK

La frase POKE I,J agisce così: memorizza il valore specificato dal secondo argomento (J) nella posizione data dal primo argomento (I). Il valore specificato da J deve essere compreso tra 0 e 255. L'indirizzo specificato da I deve essere compreso tra 0 e 65535. Questa frase va usata con attenzione perchè modifica il contenuto di determinate posizioni (I) di memoria.

La frase PEEK(I) legge il contenuto dell'indirizzo di memoria I. Con la frase PRINT PEEK(1320) si visualizza il contenuto del byte 1320. Con la frase A = PEEK(5000) si pone nella variabile A il contenuto del byte 5000.

4.6. Le funzioni del BASIC

Il tasto Π (PGRECO) corrisponde al valore 3.14159265. Ogni volta che si usa questo tasto viene considerato il valore corrispondente.

All'interno del PET si ha un orologio in tempo reale che viene aggiornato ogni 60esimo di secondo in modo da dare un orologio per 24 ore, con azzeramento alle ore 24. Il programmatore può comunicare con l'orologio mediante due funzioni che sono TI e TI$. Quando si accende il PET TI=0 e TI$=0. Il programmatore può caricare dei valori sia in TI che in TI$ anche con delle istruzioni dirette. TI viene incrementato di 1 ogni 60esimo di secondo. Il programmatore può memorizzare il valore di TI in una variabile, per esempio A, all'inizio di una serie di operazioni, poi memorizzare TI in un'altra variabile, per esempio B, alla fine delle operazioni; la differenza B-A dà il numero degli intervalli in 60esimi di secondo che sono occorsi per quel calcolo. TI$ conserva l'ora nella forma ORE-MINUTI-SECONDI (hhmmss) in una stringa di 6 caratteri. Se sono passati 10 minuti dall'una del pomeriggio, il valore di TI$ è 131000. Se vogliamo caricare nell'orologio le ore 2 e 45 minuti e 30 secondi, dobbiamo scrivere: TI$="144530".

Nel BASIC del PET si hanno altre funzioni che elenchiamo:

ABS(X) dà il valore assoluto dell'espressione X.

INT(X) dà il più grande numero intero uguale o minore di X.

RND(X) genera un numero a caso tra 0 e 1. L'argomento X ha questo effetto:

se X è negativo ogni volta che si usa RND si genera lo stesso numero a caso,
se X = 0 inizia un'altra serie di numeri a caso,
se X è magiore di 0, si ha una nuova serie di numeri a caso.

SGN(X) dà 1 se $X > 0$; dà 0 se X=0; dà —1 se $X < 0$

SIN(X) si ottiene il seno dell'espressione X interpretato in radianti. Ricordate che 1 radiante = 180/Π gradi = 57.2958 gradi. Se volete calcolare il seno di X gradi, dovete scrivere SIN(X/57.2958).

SQR(X) si ottiene la radice quadrata di X; se X è negativo si ha errore.

TAB(I) si ottiene il posizionamento in fase di stampa; si usa con la frase PRINT. Zero è la posizione più a sinistra.

ATN(X) si ottiene l'arcotangente di X espresso in radianti tra — Π/2 e Π/2.

COS(X) si ottiene il coseno di X interpretato in radianti.

EXP(X) si ottiene la costante E=2.71828 elevata a X; X non può superare 88.

FRE(X) si ottiene il numero dei bytes liberi.

LOG(X) si ottiene il logaritmo naturale (a base E) di X dove X non deve essere negativo.

SPC(I) lascia I spazi bianchi sullo schermo; si può usare nelle PRINT (I massimo 255).

TAN(X) si ottiene la tangente di X espresso in radianti.

Tutte le funzioni elencate, salvo TAB ed SPC, possono essere usate nelle frasi BASIC come variabili e vengono dal BASIC sostituite con il valore della funzione calcolata in base all'argomento.

4.7. Le stringhe del BASIC

Ricordiamo che una variabile contiene una stringa quando termina con $ (dollaro). La lunghezza massima delle stringhe è di 255 (si era detto 80 riferendosi alle possibilità dell'INPUT) caratteri; si può memorizzre una stringa di 255 caratteri concatenando con l'operatore + due stringhe o più stringhe. Infatti si provi:

```
10 A$ = "ABCDEFGHILM"
20 B$ = "NOPQRSTUVWXYZ"
30 C$ = A$ + B$
40 PRINT C$
```

e si otterrà:

```
ABCDEFGHILMNOPQRSTUVWXYZ
```

Le stringhe possono essere confrontate tra loro come le variabili numeriche; infatti la codifica interna dei caratteri è tale da consentire l'ordinamento alfabetico. Provate a scrivere un programma che confronti e metta in ordine tutti i caratteri del PET e poi portateli sul video per controllare l'ordinamento. Per le variabili stringa valgono quindi tutti gli operatori relazionali già visti, e cioè: diverso, uguale, maggiore e minore. Provate questo semplicissimo programma:

```
10 INPUT“SCRIVI UNA STRINGA A$”; A$
20 INPUT“SCRIVI UNA STRINGA B$”; B$
30 IF A$=B$ THEN PRINT”A$=B$”,A$,B$: GOTO 60
40 IF A$<B$ THEN PRINT“ A$<B$”,A$,B$: GOTO 60
50 PRINT “A$>B$”,A$,B$
60 PRINT “SE VUOI TERMINARE SCRIVI PER A$ 99”
70 IF A$<>“99” GOTO 10
80 STOP
```

4.8. Le funzioni di stringa

Elenchiamo le funzioni BASIC che operano sulle stringhe.

STR$(X)	dà la rappresentazione di X, che deve essere un numero, sotto forma di stringa; se il numero è positivo la stringa inizia con uno spazio; se negativo con il segno meno.
CHR$(N)	dà il carattere ASCII che corrisponde al codice numerico fornito per N. A corrisponde al numero 65, mentre B corrisponde al numero 66. Se scrivete: PRINT CHR$(65)+CHR$(66) ottenete AB
ASC(X$)	dà di ritorno il valore numerico ASCII del primo carattere della stringa X$.
LEFT$(X$,I)	si ottengono gli I caratteri più a sinistra della stringa X$.
LEN(X$)	si ottiene la lunghezza in caratteri della stringa X$.
MID$(X$,I,J)	ritorna J caratteri della stringa X$ a partire dalla posizione I.
MID$(X$,I)	ritornano i caratteri di X$ a partire dalla posizione I.
RIGHT$(X$,I)	si ottengono gli I caratteri più a destra della stringa X$.
VAL(X$)	ritorna la rappresentazione numerica della stringa; se X$ non è numerica dà zero.

Se si adoperano le funzioni di cui sopra con argomenti che non rispettano le caratteristiche delle stringhe, si ha errore. Ricordiamo che quando si caricano stringhe con la frase DATA, non si devono usare le virgolette per delimitare le stringhe, a meno che non si vogliano inserire spazi all'inizio o alla fine. Valgono le stesse regole viste per la frase INPUT.

4.9. I sottoprogrammi

Durante la stesura di un programma può verificarsi, in diversi punti del programma stesso, la necessità di ripetere la stessa serie di istruzioni. Per evitare di allungare inutilmente il programma, si possono raggruppare le istruzioni da ripetere più volte in un solo punto del programma (cioè da una determinata linea ad un'altra determinata linea) e poi, servendosi di apposite frasi BASIC, saltare ad eseguire la sequenza desiderata, ritornando, alla fine della sequenza, al punto di partenza. Chiariamo con un esempio. Si debba ripetere questa serie di istruzioni:

```
LET C=A+B—(D/5)
LET E=A—(D+B)/6
LET F=(E+C)/8
```

Supponiamo che la serie di istruzioni prima elencate debba essere ripetuta in 3 diversi punti del programma. Allora scriviamo così:

```
10   .......
20   .......
30   GOSUB 1000
40   PRINTC,A,F
50   .......
60   .......
70   ......
80   GOSUB 1000
90   .......
100  .......
110  .......
120  .......
130  .......
140  .......
150  .......
160  GOSUB 1000
170  .......
180  .......
190  STOP
1000 REM SOTTOPROGRAMMA COMUNE
1010 LET C=A+B—(D/5)
1020 LET E=A—(D+B)/6
1030 LET F=(E+C)/8
1040 RETURN
```

Alla linea 30 il programma esegue un GOSUB che significa *salto a sottoprogramma* e va alla linea 1000. Il sistema *ricorda* che il GOSUB è avvenuto dalla linea 30 ed al termine del sottoprogramma, cioè quando

incontra RETURN, ritorna alla linea 40 (quella dopo la 30). Ancora alla linea 80 il programma fa GOSUB 1000 ed alla linea 1040 con il RETURN torna alla linea 90 (quella dopo 80). Ancora alla linea 160 il programma fa GOSUB 1000 alla ed alla linea 1040 ritorna alla 170 (quella dopo 160). Riepilogando possiamo dire:

- la frase per salto a sottoprogramma è la GOSUB numero-linea;
- il sottoprogramma inizia con qualunque frase BASIC, ma deve *logicamente* terminare con RETURN.

L'uso dei sottoprogrammi permette di scrivere dei buoni programmi, ma si deve fare attenzione a non dimenticare la frase RETURN. È possibile che dall'interno di un sottoprogramma si salti ad un altro sottoprogramma, cioè è possibile quello che viene chiamato l'annidamento dei sottoprogrammi. Per saltare ad un sottoprogramma si può usare anche il *salto calcolato,* che è simile all'ON ... GOTO. Possiamo scrivere: ON variabile GOSUB n1,n2,n3 .... cioè a seconda del valore di variabile si salterà ad uno o ad un altro sottoprogramma. Se variabile=1 si salta al sottoprogramma che inizia alla linea n1, ecc.

4.10. Le funzioni definite dall'utente

Quando un sottoprogramma può essere risolto solo in una linea di programma con un calcolo del tipo: variabile = espressione, si ricorre alla definizione di una funzione utente. Si opera così:

DEF FN variabile(X) = espressione

La variabile che segue FN deve essere di tipo numerico; la X è l'argomento della funzione e l'espressione dopo l'uguale può essere qualunque espressione consentita dal BASIC (può anche richiamare altre funzioni del BASIC stesso). La X si chiama *argomento dummy,* nel senso che, quando si richiama la funzione nel programma, si usa l'argomento che serve al momento e che diventa la variabile con la quale vengono svolti i calcoli indicati. Le funzioni definite dall'utente si usano nel programma come le funzioni proprie del BASIC. È buona regola definire le funzioni all'inizio del programma.

4.11. Precisazioni sulle variabili con indice

Abbiamo già trattato le variabili con indice, nel caso di *un solo* indice. Se la variabile con indice ha un solo indice e meno di 10 elementi, non è necessario definirla con una frase DIM; la definisce implicitamente il sistema. Se le variabili hanno più di un indice si deve usare la DIM. Se si scrive DIM A(9,8), si è definita una variabile che ha (9+1)*(8+1)=90 elementi. Il numero massimo di elementi che può avere una variabile con indici non è definito a priori, ma dipende dalla memoria disponibile, mentre

il numero di indici possibili non deve superare 255. Si veda l'Appendice C. Ricordiamo che gli indici partono da zero.

Si usa chiamare *matrice* una variabile con indici che abbia più di un indice. Se la matrice è a due dimensioni tipo A(I,K), I si dice indice di riga e K indice di colonna. L'uso delle variabili con indice unito all'uso delle frasi FOR-NEXT permette di scrivere facilmente programmi che trattano dati organizzati in gruppi.

4.12. Gli operatori logici

Esistono 3 operatori logici: AND, OR, NOT, si chiamano anche operatori booleani, il cui significato, chiarito da esempi, è:

AND se scriviamo 2 IF A<5 AND B<7 THEN 7
3 ...
opera così: se le due espressioni A<5 e B<7 sono ambedue vere, cioè si verificano contemporaneamente il programma prosegue dalla linea 7, se ambedue o una delle due non è vera, il programma prosegue dalla linea 3.

OR se scriviamo 5 IF A<1 OR B<2 THEN 8
6 ...
opera così: se una delle due espressioni A<1 e B<2 è vera o se ambedue sono vere, il programma prosegue dalla linea 8, se ambedue false prosegue dalla linea 6.

NOT se scriviamo 2 IF NOT Q3 THEN 4
3 ...
4 ...
opera così: se NOT Q3 è vera (questo si verifica se Q3 è falso) allora il programma salta alla linea 4, se NOT Q3 è falso salta alla seguente, cioè alla 3.

Gli operatori logici lavorano a livello di bit sui dati chiamati in causa.

Si possono tracciare le tabelle della verità per gli operatori logici così:

AND

A	B	RISULTATO
1	1	1
0	1	0
1	0	0
0	0	0

OR

A	B	RISULTATO
1	1	1
0	1	1
1	0	1
0	0	0

NOT

A	RISULTATO
1	0
0	1

Gli operatori logici si possono usare anche nelle espressioni tipo:

```
10 X% = A% AND B%
20 PRINT X%
```

se A = 7 e B = 11 dobbiamo considerare cosa sta nella memoria del calcolatore in binario

$A = (0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 111)_2 = (7)_{10}$
$B = (0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 10\ 11)_2 = (11)_{10}$
risulta $X = (0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 11)_2 = (3)_{10}$

conformemente alle tabelle della verità.

4.13. La frase WAIT

È una frase di attesa, opera così:

WAIT I,J,K legge lo stato della locazione I
esegue l'OR esclusivo con K
poi effettua l'AND con J fino ad un risultato diverso da zero Se K viene omesso, è assunto uguale a zero.
Serve per creare delle attese nel programma e non può essere interrotto da STOP.

I deve essere compreso tra 0 e 65535, J e K devono essere compresi tra 0 e 255 la tabella della verità per XOR è:

XOR

A	B	RISULTATO
T	T	F
T	F	F
F	T	T
F	F	F

4.14. La frase GET

Si scrive GET variabile ed accetta un singolo carattere dalla tastiera senza aspettare che sia premuto il RETURN.

10 GET A accetta un carattere numerico dalla tastiera, se nessun carattere è premuto A = 0
10 GET A$ accetta un carattere stringa da tastiera. Se si scrive:

10 GET A$
20 IF A$ = ""THEN 10 crea un attesa nel programma fino a quando viene schiacciato un tasto, infatti se non viene schiacciato alcun tasto A$ = alla stringa nulla.

4.15. Istruzioni multiple

Se si vogliono scrivere più istruzioni sulla stessa linea, si può farlo separando le istruzioni con i *due punti* (:). Esempio:

10 IF A=B THEN PRINT "A=B": GOTO 100

4.16. L'uso del punto interrogativo

Si può usare il punto interrogativo al posto della parola PRINT per il video, comunque se il programma viene richiesto in stampa con LIST si vede PRINT dove si era messo?.

4.17. Come ripartire in fase prova programma

Nel paragrafo 3.3. si è visto come operare per scrivere un programma in memoria e per eseguirlo; riepiloghiamo:

- Il PET dà READY,
- Il programmatore scrive NEW,
- Il programmatore scrive il programma BASIC,
- Il programmatore scrive RUN per provare il programma

Se tutto va bene si ottengono dei risultati. Il programmatore può apportare le necessarie correzioni e riprovare il programma. Ricordiamo, anche che:

- un'operazione di INPUT non può essere interrotta, non sente lo STOP,
- scrivere RUN fa ricominciare l'esecuzione del programma, ma riinizializza le variabili e quindi perde i dati precedentemente calcolati.

Se si desidera fare ripartire un programma da una determinata linea, si può scrivere in modo diretto GOTO num-linea; con questo tipo di ripartenza non vengono riinizializzate le variabili. Se si ricomincia comunque a scrivere un programma senza dare NEW, le vecchie istruzioni, se non sostituite da altre con lo stesso numero di linea, rimangono e quindi si ottiene un piccolo pasticcio. Se proprio non si sa più cosa fare, si può spegnere il calcolatore e riaccenderlo, chiaramente perdendo però quanto c'è in memoria.

4.18. Esercizi riepilogativi

A conclusione dello studio dei primi 4 capitoli, si consiglia di svolgere gli

esercizi che seguono con il metodo corretto, e cioè:

- stesura di una breve analisi,
- stesura del diagramma a blocchi,
- codifica del programma,
- preparazione dei casi da provare,
- prova del programma con conseguente eventuale correzione,
- memorizzazione del programma su cassetta,
- prova di richiamo del programma da cassetta,
- accodamento di programmi su cassetta.

Esercizio 9: Leggere dalla tastiera 50 numeri decimali e memorizzarli in un vettore N. Calcolare le due medie MP e MD, rispettivamente dei numeri che occupano nel vettore posizioni pari e posizioni dispari. Stampare il vettore N dei 50 numeri e le due medie calcolate.

Esercizio 10: Scrivere un programma che servendosi della funzione BASIC RND(X) generi tre sequenze di numeri a caso di 50 numeri ciascuna, usando per la prima sequenza X negativo, per la seconda X = 0 e per la terza X positivo. Ogni sequenza va evidenziata allo schermo, facendola precedere da una frase di commento.

Esercizio 11: Determinare quanti giorni intercorrono tra due date che appartengono allo stesso anno. Il formato delle due date è: GGMMAA.

Esercizio 12: Scrivere un programma che faccia la somma dei primi 500 numeri interi naturali. Servendosi dell'orologio TI del sistema, calcolare quanto tempo si impiega a fare tale somma. Provare a fare tale calcolo servendosi anche di TI$.

Esercizio 13: Scrivere un programma che calcoli la tavola Pitagorica per i numeri da 1 a 12.

Esercizio 14: Scrivere un programma per generare un triangolo di Tartaglia di 10 righe.

Il risultato deve essere:

```
1
1  1
1  2   1
1  3   3   1
1  4   6   4   1
1  5  10  10   5    1
1  6  15  20  15    6   1
1  7  21  35  35   21   7   1
1  8  28  56  70   56  28   8  1
1  9  36  84  126 126  84  36  9  1
```

Esercizio 15: Scrivere un programma che consenta di convertire un numero di 9 cifre (massimo 999.999.999) in lettere. (Questo problema si presenta quando si devono stampare degli assegni).

Esercizio 16: Provate a scrivere un programma che, servendosi dei caratteri grafici, tracci un disegno sullo schermo. Ricordate che per attivare i tasti grafici, se usate il primo set di caratteri, basta tenere premuto il tasto SHIFT. Il secondo set di caratteri non ha tasti grafici, ma solo minuscole e maiuscole.

CAPITOLO 5

STRUTTURA DEL PET

5.1. Alcune notizie sulla struttura del calcolatore

Al centro del PET c'è un Microprocessore MCS 6502; esso:

- controlla totalmente le operazioni dello schermo,
- controlla le operazioni della tastiera,
- controlla le operazioni della cassetta,
- controlla le periferiche addizionali che possono essere aggiunte.

Il sistema operativo del PET risiede su memoria *ROM* (Read-Only-Memory - memoria a sola lettura) e quindi non può essere distrutto dall'utente. La memoria ROM è formata da 14 k, dove k = 1024 e si riferisce ai bytes che sono insiemi di 8 bits. Ad ogni byte della memoria è assegnato un indirizzo, cioè un numero di riferimento. La memoria nella quale l'utente può leggere e scrivere si chiama *RAM* (Random Access Memory - memoria ad accesso casuale).

Oltre alla ROM del sistema operativo, esistono nel PET altri 3 tipi di memoria e precisamente:

- Memoria RAM di lettura/scrittura per l'utente
- Memoria RAM di schermo
- Memoria speciale di Input/Output (comunicazione con l'esterno).

Il microprocessore 6502 comunica con le memorie attraverso un canale chiamato *BUS*, come esemplificato dallo schema di figura 5.1.

La ROM permette al PET di eseguire le sue operazioni; essa contiene una serie di programmi scritti dalla COMMODORE che fanno:

- la scansione della tastiera,
- realizzano la scrittura sullo schermo,
- controllano l'INPUT,
- realizzano orologi in tempo reale,
- eseguono i comandi che l'utente batte sulla tastiera.

Le memorie a sola lettura costano meno delle altre e permettono di dare la massima sicurezza e la più alta velocità operativa. Il *Sistema Operativo* (corredo di programmi del calcolatore) è indistruttibile dalla tastiera e consente all'utente di conversare in BASIC con il PET.

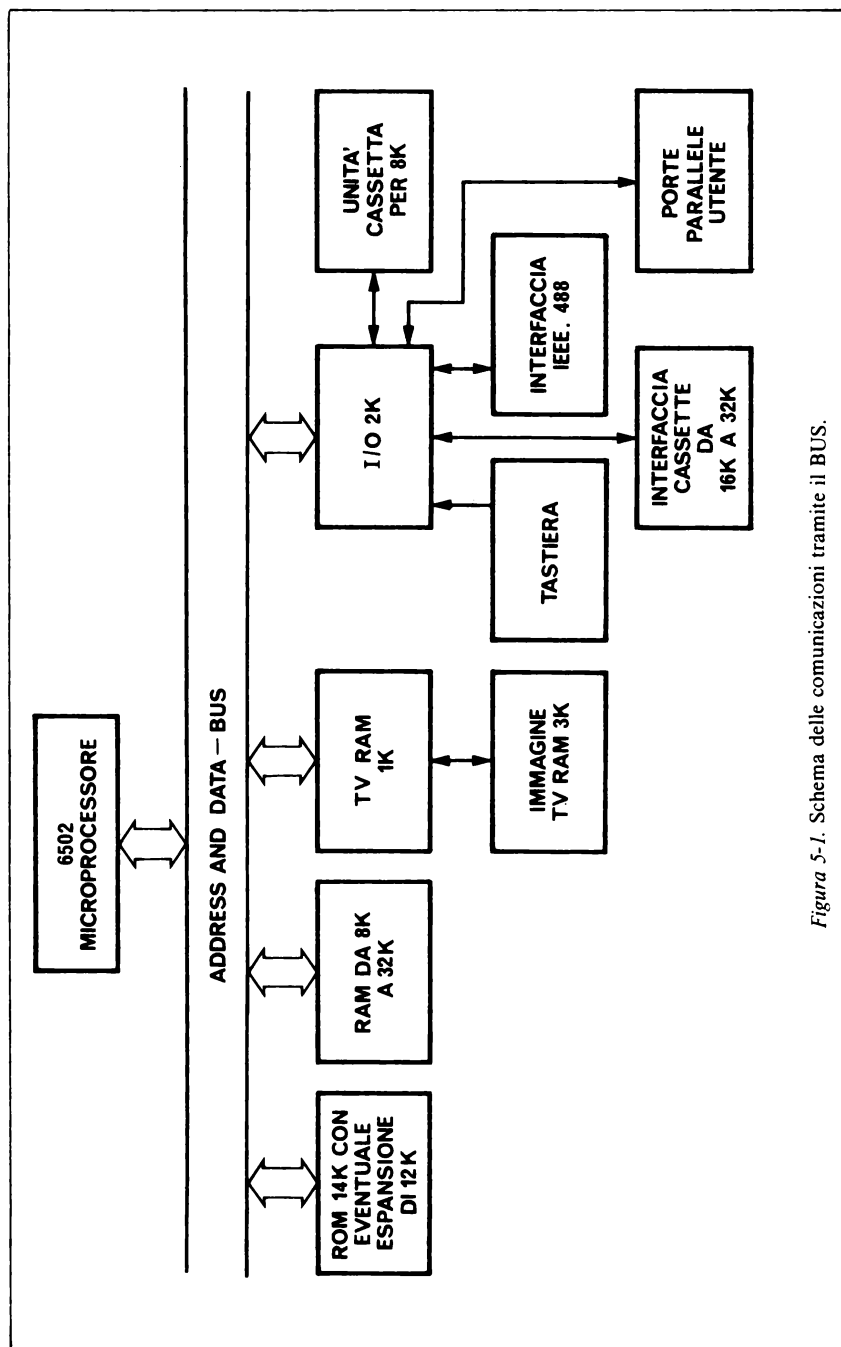



Figura 5-1. Schema delle comunicazioni tramite il BUS.

La *memoria di I/O* (input/output) contiene dei dispositivi per l'input e l'output, chiamati "PIA E VIA" che consentono di controllare i singoli bit che vengono manipolati. Queste locazioni vengono usate direttamente dal sistema operativo.

La *memoria TV* viene continuamente scandita dal circuito di controllo del tubo a raggi catodici che estrae i bytes e li usa per indirizzarsi ad uno speciale generatore di carattere nella ROM e fa apparire i caratteri sullo schermo. Questo modo di funzionare è automatico ed a carico del sistema.

La *memoria RAM dell'utente* viene usata per memorizzare i programmi dell'utente in fase operativa.

I caratteri vengono rappresentati nella memoria del PET usando i caratteri ASCII standard; tali caratteri usano 7 bit, l'ottavo bit indica comandi BASIC o i caratteri grafici. Per esempio: A è rappresentato da 0 10 0 0 0 0 1, mentre la picca è rappresentata da 110 0 0 0 0 1.

La memoria dello schermo non può rappresentare tutti i caratteri che possono essere conservati in memoria RAM, ma solo 64 caratteri, che sono poi quelli disponibili sulla tastiera, non considerando i tasti dei comandi. La codifica della memoria di schermo è diversa da quella della memoria principale, ma la conversione viene fatta dal sistema. La memoria di schermo inizia all'indirizzo $8000.

5.2. Uso della memoria di schermo

Per inserire dati nella memoria di schermo, si può procedere in 3 modi:

1. se la tastiera è abilitata, il carattere premuto sulla tastiera compare sullo schermo;
2. se si usa il comando PRINT seguito o da una lista di variabili o da stringhe tra virgolette o da numeri sullo schermo compare o il contenuto delle variabili o compaiono le stringhe senza virgolette, o i numeri;
3. scrivendo con la POKE in uno degli indirizzi della memoria di schermo un carattere, questo compare sullo schermo (questo modo è meno usuale).

L'indirizzo $8000 corrisponde alla posizione in alto a sinistra. Si può ricorrere al terzo modo se si vuole che la modifica della memoria di schermo sia più veloce. Infatti il sistema modifica la memoria di schermo solo negli istanti in cui essa non viene usata per far apparire il carattere e questo rallenta la velocità di modifica della stessa. Il *cursore* lampeggiante è una indicazione visuale all'utente della successiva posizione sulla memoria di schermo. Il lampeggiamento viene ottenuto dal sistema così: lo schermo viene riciclato circa ogni sessantesimo di secondo e si ha un interrupt. L'interrupt serve a posizionare un contatore, quando questo raggiunge il valore 30 il carattere individuato viene ribaltato sull'ottavo bit, cioè viene

data la rappresentazione in campo inverso. L'alternarsi delle due rappresentazioni dà l'effetto di lampeggiamento.

5.3. Mappa della memoria

Nello schema che segue riportiamo la mappa della memoria per i diversi sistemi PET. Come si può vedere, il primo k di memoria serve come area di

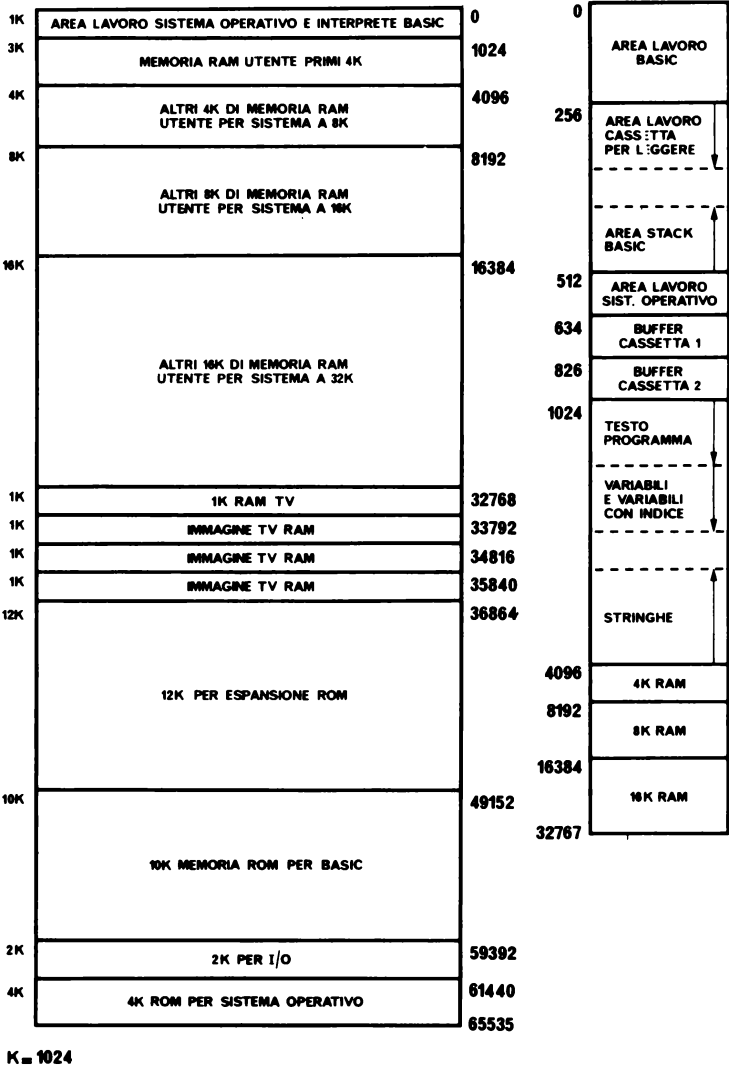



Figura 5-2. Mappa della memoria in byte.

lavoro per il BASIC ed il sistema operativo. Gli altri k di memoria RAM (fino a 32767 per i sistemi a 32 k) servono per i programi dell'utente; naturalmente quando si dice programmi, si intendono: istruzioni, aree di lavoro e costanti di lavoro. All'indirizzo 32768 ($8000, cioè 8000 in esadecimale) inizia la memoria di schermo. Dall'indirizzo 49152 all'indirizzo 59391 è localizzata l'area ROM usata dall'interprete BASIC. Dall'indirizzo 59392 all'indirizzo 61439 è localizzata l'area usata dal sistema per le operazioni di I/O. Infine, negli ultimi 2 k di memoria, indirizzi più alti, è localizzato il Sistema Operativo, sempre in memoria ROM.

Riportiamo anche una mappa dell'utilizzo della memoria RAM. Le due zone di buffer per le 2 possibili cassette, possono essere usate dall'utente in assenza delle cassette. Nell'area RAM utente è schematizzato l'utilizzo della memoria da parte dei programmi in BASIC.

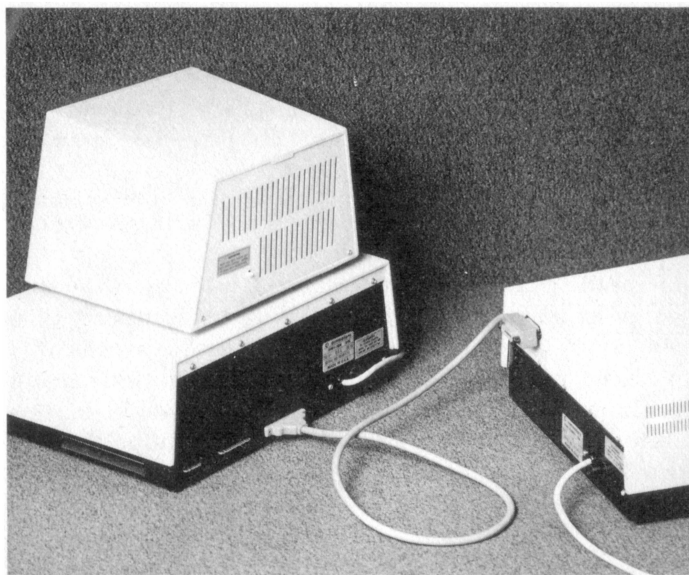



Figura 5-3. Connessione microcomputer-unità floppy disk.

5.4. Registrazione su cassetta

I dati sulla cassetta sono registrati due volte in due blocchi consecutivi a due diverse frequenze. Nel secondo blocco è anche registrata una somma di controllo a livello di bit. In tal modo è possibile avere una buona sicurezza nei dati registrati. In fase di lettura i due blocchi vengono letti e controllati.

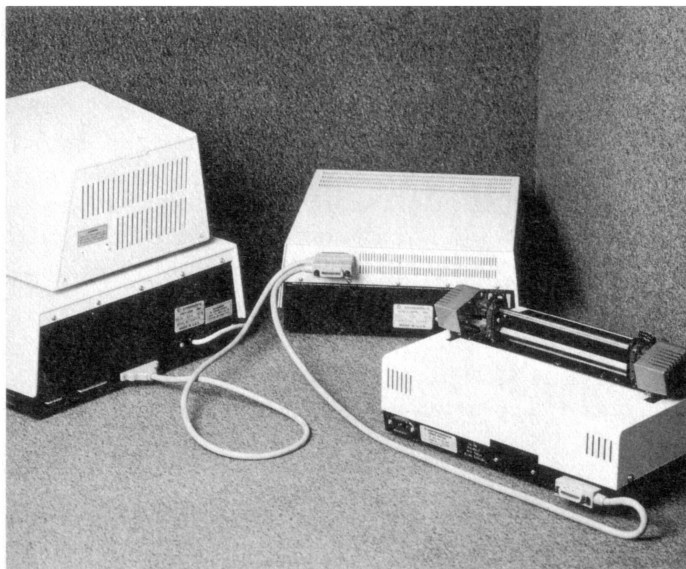



Figura 5-4. Connessione microcomputer - unità floppy disk - stampante.

Dato che le unità a nastro registrano a velocità variabile, vengono usate delle tecniche di sincronizzazione che permettono di riconoscere l'inizio e la fine dei singoli blocchi.

Il primo carattere di ciascun blocco abilita il sistema a riconoscere il tipo del blocco: programmi, dati, inizio file, fine file, fine nastro. I blocchi su nastro sono lunghi 192 caratteri ed il sistema li gestisce usando un buffer di 192 caratteri che inizia all'indirizzo decimale 634. Il programmatore usa il nastro senza preoccuparsi del buffer fisico per i blocchi, cioè usa il file secondo la logica del suo programma ed il sistema gestisce l'INPUT/OUTPUT fisico sul nastro. I programmi quando vengono scritti sul nastro recano anche l'indicazione del loro indirizzo di inizio.

CAPITOLO 6

TRATTAMENTO DEI FILES

6.1. Identificazione delle periferiche

Tutte le periferiche con cui il PET comunica, vengono identificate attraverso un numero intero. L'*assegnazione dei numeri* alle periferiche è di norma la seguente:

0 corrisponde alla TASTIERA
1 corrisponde alla CASSETTA 1
2 corrisponde alla CASSETTA 2
3 corrisponde al VIDEO
4 corrisponde alla STAMPANTE TIPO 2022 e 2023
5 corrisponde alla STAMPANTE TIPO LINA 20
8 corrisponde all'UNITA' A DISCHI

gli altri numeri 6, 7 e da 9 a 30 sono per unità collegate all'interfaccia IEEE o alle porte parallele.

La *stampante* ed il *disco* sono dispositivi IEEE-4888; a questi dispositivi competono numeri da 4 a 30. Eccetto casi speciali questi numeri vengono fissati al momento dell'installazione delle apparecchiatura. Su alcuni dispositivi la codifica del numero di riconoscimento viene fatta con interruttori, su altri con connessioni saldate.

6.2. I files

Le registrazioni che il sistema fa sulle periferiche, si chiamano *files.* Queste registrazioni, in alcuni casi, possono essere rilette dal sistema. Si hanno quindi files di INPUT e files di OUTPUT. I files che non possono essere letti dal sistema sono queli diretti alla stampante ed al video. I files che possono essere trattati in INPUT ed in OUTPUT sono quelli registrati sulle cassette e sui dischi. I files sono composti da *records,* ed i records sono composti da campi di diversa natura per quantità e qualità dei caratteri.

Si osservi lo schema che segue:

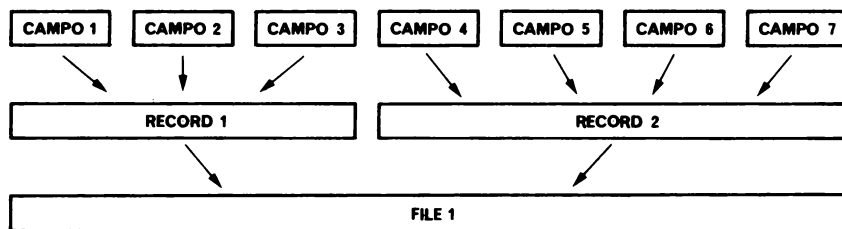



Figura 6-1. Schema FILE - RECORD - CAMPO.

Un file è quindi una collezione di records. Se consideriamo un file di stampa, per esempio, i records sono le diverse righe che si stampano. Alcune righe hanno lo stesso tracciato, cioè sono formate dagli stessi tipi di campi, mentre altre hanno tracciati diversi. Un file può essere formato da records tutti di formato uguale, ma anche da records di formato diverso che si alternano tra loro.

Il sistema considera qualunque dispositivo al quale possa mandare dei dati o dal quale possa ricevere dei dati come un file. Chiamiamo il file in questione *file logico.* Il sistema quindi lavora con dei files logici. Ogni file logico, con cui il sistema lavora, viene contraddistinto da un numero, che si chiama *numero logico del file.* Il sistema può lavorare contemporaneamente al massimo con 10 files logici. Però il numero logico del file può essere un numero intero compreso tra 1 e 255. Non si deve confondere il numero identificatore della periferica con il numero logico del file; si tratta di due concetti diversi.

Un file, oltre ad essere riconosciuto da un programma con un numero logico, in certi casi, e precisamente nel caso di registrazioni su supporti magnetici, è dotato di un *nome.* Questo significa che sui supporti magnetici ogni file è contraddistinto da una etichetta che reca il suo nome. Il nome viene assegnato al momento della creazione del file e quindi i programmi che devono elaborare quel file, devono conoscerne il nome per poterlo richiamare. Per chiarire ulteriormente possiamo dire che il nome resta appiccicato al file, mentre il numero logico può cambiare da programma a programma. Inoltre, in certi casi, per trattare i files bisogna riferirsi ad un parametro che si chiama *indirizzo secondario* e di cui vedremo più avanti il significato.

Riepilogando: per poter studiare le istruzioni BASIC che trattano i files dobbiamo aver presenti questi 4 parametri:

- numero logico del file che abbreviamo in LF
- numero del dispositivo che abbreviamo in D
- indirizzo secondario che abbreviamo in SA
- nome del file ch abbreviamo in FN.

6.3. Comandi BASIC per i files

Fino ad ora abbiamo usato per l'INPUT/OUTPUT i comandi INPUT e PRINT, e precisamente INPUT da tastiera e PRINT sullo schermo. Ora studieremo dettagliatamente i seguenti comandi:

- OPEN	serve per aprire un file; se il file non è aperto non si possono dare comandi di lettura o scrittura;
- CLOSE	serve per chiudere un file; quando un file non serve più, esso deve essere chiuso, cioè escluso dalla lavorazione;
- PRINT #	scrive sul dispositivo;
- CMD	ha lo stesso significato del PRINT, ma lascia i dispostivo attivo sul bus anche dopo l'esecuzione del comando;
- INPUT #	legge dal dispositivo;
- GET #	legge un carattere dal dispositivo.

Esamineremo dettagliatamente ogni comando precisando come si devono usare i 4 parametri visti nel precedente paragrafo e cioè: LF,D,SA,FN. Per quanto riguarda questi 4 parametri, si sono già visti i possibili valori per LF e D. Chiariamo le carateristiche di SA e di FN.
SA (indirizzo secondario) permette di far funzionare in diversi modi una periferica. Questo significa che con un indirizzo secondario, la periferica compie l'operazione di lettura, mentre con un altro indirizzo secondario compie l'operazione di scrittura. I numeri che si possono usare come indirizzi secondari, sono fissati secondo convenzioni per parecchi dispositivi e li vedremo più avanti. Per quanto riguarda il nome del file, FN, esso può essere una stringa fino a 128 caratteri, tale nome identifica il file sul dispositivo.

6.4. Apertura dei files

Per informare il BASIC su quale file e come deve lavorare, si deve aprire il file con la frase OPEN. Il formato generale della frase è:

OPEN LF,D,SA,FN

dove i quattro parametri hanno il significato visto nei paragrafi precedenti. Non è sempre necessario usare tutti i 4 parametri. Il primo, LF, ci vuole sempre, se D manca, viene considerato uguale a 1 e quindi il sistema

considera come dispositivo la prima cassetta. I parametri LF,D e SA possono essere sia delle costanti che delle variabili e quindi essere calcolati dal programma o letti dall'esterno prima di aprire il file. Il nome del file è necessario per i file su disco ed è consigliabile usarlo sempre anche per i files su nastro.

La frase OPEN 1,2,1 è interpretata così:

- LF : viene aperto il file logico numero 1.
- D : il file logico 1 è assegnato al dispositivo numero 2, che sarebbe la seconda cassetta.
- SA : l'indirizzo secondario 1 significa che si vuole scrivere sul nastro.
- FN : a questo file non è stato assegnato un nome.

La frase OPEN 3 è interpretata così:

- LF : viene aperto il file logico numero 3.
- D : manca e quindi viene preso uguale ad 1 ed è aperto il file 3 sul dispositivo 1, la prima casetta.
- A : manca e quindi viene preso uguale a 0 (zero); l'indirizzo secondario zero vuol dire operazione di lettura da nastro.
- FN : manca il nome del file.

La frase OPEN 12,4,1 è interpretata così:

- LF : viene aperto il file logico numero 12.
- D : il file 12 viene assegnato alla stampante 4.
- SA : l'indirizzo secondario 1 riferito ad una stampante significa che si vuole stampare sotto controllo di un formato.
- FN :manca e non ci può essere per la stampante.

Vediamo quali sono i valori possibili per SA per le periferiche più comuni:

Indirizzi secondari SA per la *stampante:*

- 0 Stampa normale.
- 1 Stampa sotto controllo di un formato predisposto.
- 2 Predisposizione di un formato per la stampa dei dati.
- 3 Dà la possibilità di variare il numero di linee per pagina.
- 4 Attiva l'uso dei messaggi diagnostici estesi.
- 5 Permette di programmare un carattere di stampa personale.
- 6 Vale solo per il modello 2022. Permette di predisporre la spaziatura tra le linee.

Indirizzi secondari SA per la *cassetta:*

- 0 Operazione di lettura.
- 1 Operazione di scrittura.
- 2 Operazione di scrittura con scrittura sul nastro di un contrassegno di fine nastro (END OF TAPE).

Gli indirizzi secondari per i dischi verranno studiati più avanti. Ricordiamo che se manca SA esso viene considerato *zero.*

Nel paragrafo 3.8. abbiamo visto come si possono registrare programmi sulla cassetta, verificare se la registrazione è buona e leggere in memoria programmi dalla cassetta. Anche un programma registrato su cassetta è un file, solo che i comandi LOAD, VERIFY e SAVE non necessitano prima della OPEN.

6.5. Uso dei file su nastro magnetico (cassetta)

I files su nastro possono essere aperti per due scopi distinti:

- per scrivere dal PET sul nastro,
- per leggere dal nastro ed inviare dati al PET.

Nel diagramma che segue si mettono in evidenza le operazioni che vengono fatte per aprire un file per scrivere. Dal primo blocco si vede che il file può essere aperto per scrivere con la OPEN e con la SAVE. Si vede anche che se i tasti della cassetta PLAY & RECORD sono già schiacciati, non viene evidenziata la richiesta, e che se non si lavora sotto controllo del programma, viene richiesto il nome del file.
NOTA: OP significa che il sistema operativo riceve il controllo e lavora.

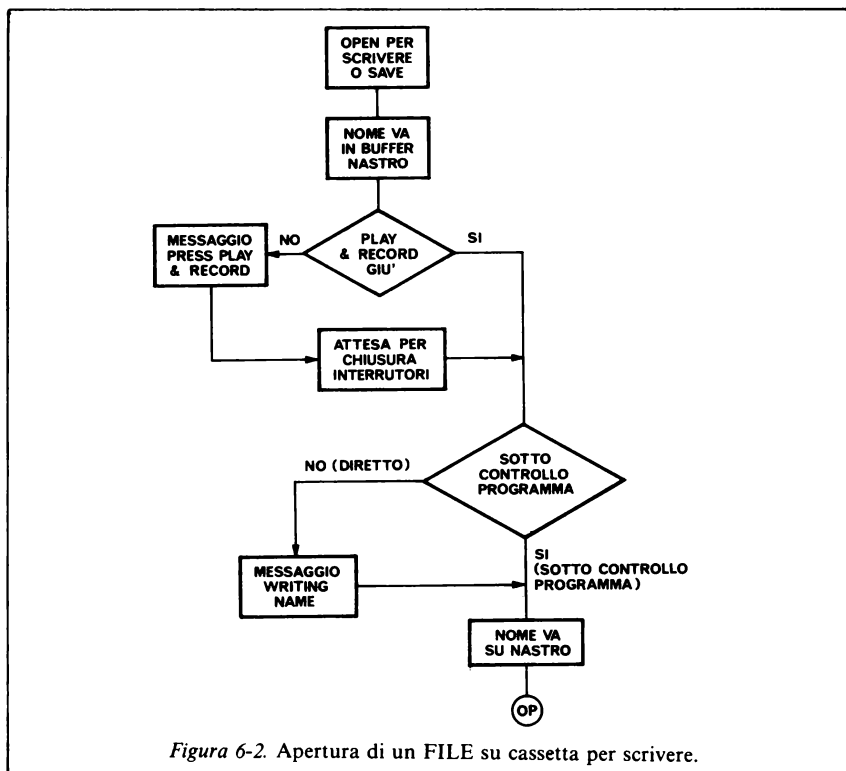



Figura 6-2. Apertura di un FILE su cassetta per scrivere.

Nel diagramma che segue vengono invece messe in evidenza le operazioni che vengono fatte per aprire un file per leggere. Dal primo blocco si vede che il file può essere aperto con la OPEN per leggere o con la LOAD. Si vede anche che si ha una diversa evidenziazione di messaggi se si lavora sotto controllo di un programma o direttamente e se il tasto PLAY è già schiacciato. Si vede anche che dopo una operazione di LOAD vengono inizializzate le variabili del BASIC.
NOTA: OP significa che prende il controllo il sistema operativo, mentre B significa che prende il controllo il BASIC.

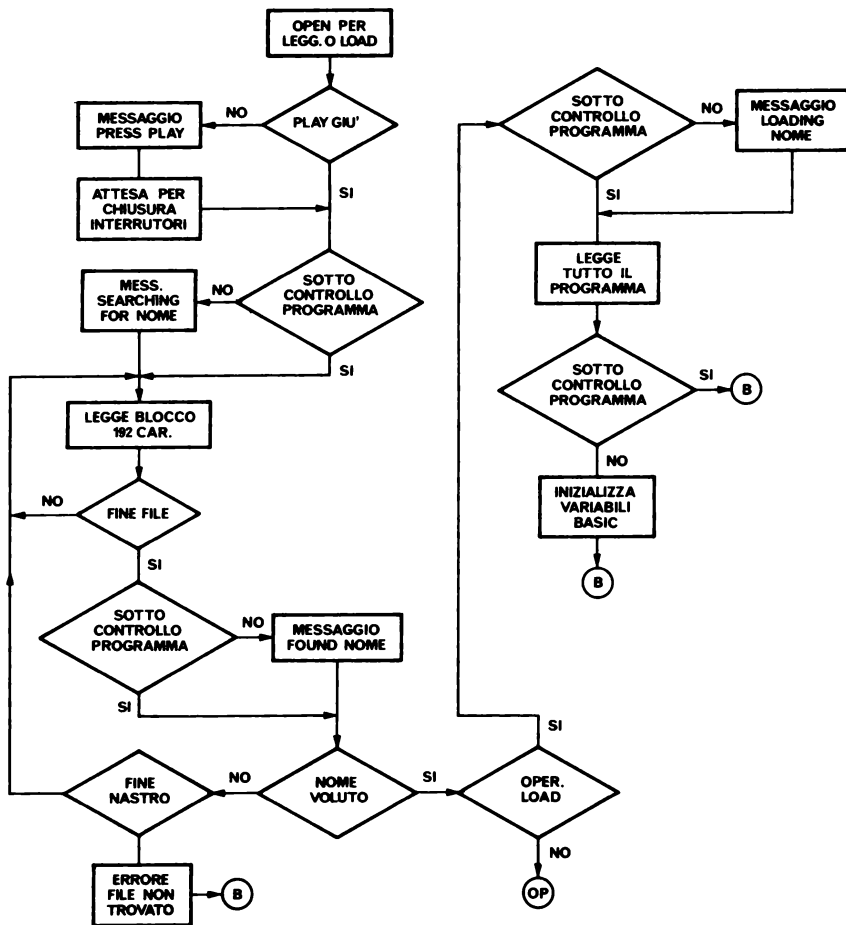



Figura 6-3. Apertura di un FILE su cassetta per leggere.

6.6. Considerazioni sulla OPEN per i dispositivi IEEE-488

Se D è maggiore o uguale a 4, il sistema considera il dispositivo del tipo IEEE-488. In questo caso, se non è specificato il nome del file, non viene inviato alcunché sul BUS. Se, invece, è specificato il nome del file, allora il sistema operativo invia sul BUS un segnale di attention verso il dispositivo selezionato insieme ad un indirizzo secondario che è un OR del numero esadecimale "F0" e del SA specificato nella OPEN. Le periferiche usano il nome del file e questo indirizzo secondario calcolato per lavorare con il file.

6.7. INPUT# Ingresso di stringhe e di variabili numeriche o alfanumeriche

Il formato della frase per leggere da un file è:

INPUT # LF, lista di variabili (stringhe o numeriche)

dove LF è il numero logico del file aperto con una OPEN e # è il carattere speciale che specifica INPUT non da tastiera. Le variabili della lista vengono riempite con dati letti dal file. Si deve fare attenzione affinchè i nomi delle variabili corrispondano esattamente al tipo di variabile che si legge; in caso contrario si ha segnalazione di errore. Se non si è sicuri della qualità dei dati che si leggono, si possono leggere tutti i dati come stringhe e poi operare con le funzioni di stringa per sistemare i dati numerici.

Esempio di lettura *da file:* si abbia un file su nastro di nome "VECTOR" che contenga 50 numeri, si vogliano leggere i 50 numeri ed evidenziarli sul video. Le istruzioni necessarie sono:

10 OPEN 1,1,0, "VECTOR"	apre il file logico 1 sulla cassetta 1 per leggere e cerca il file di nome VECTOR
20 FOR K = 1 TO 50	legge i 50 numeri uno alla volta nella variabile X
30 INPUT # 1,X	
40 PRINT X	stampa X sul video
50 NEXT K	
60 CLOSE 1	chiude il file logico 1

6.8. GET# Ingresso di un carattere

Non tutte le periferiche sono in grado di trasmettere i dati di una forma accettabile dal BASIC. Esistono infatti serie di dati binari che il BASIC non riconosce ed alcuni dispositivi IEEE-488 inviano dati di questo tipo funzionando correttamente. Inoltre si può desiderare di ricevere carattere per carattere anche dei dati che sono accettati dal BASIC. Per ricevere carattere per carattere dei dati, si usa, relativamente ai files, la frase:

GET # LF, variabile

dove LF è il numero logico del file e variabile è dove si vuole avere il

carattere. Questa istruzione si può usare sia per la cassetta che per i dispositivi IEEE-488.

6.9. Considerazioni sull'operazione di lettura da file

Quando si legge dal nastro, il sistema legge il nastro a blocchi e mette un blocco nel buffer di lettura; per ogni richiesta di dati trasferisce dal buffer nelle variabili elencate i dati richiesti, quando i dati del buffer sono terminati, legge un altro blocco. Naturalmente il programmatore non si accorge di questo lavoro che è a carico del sistema operativo. Durante la lettura può essere riscontrato un errore; in tale caso il sistema modifica opportunamente la *parola di stato,* che è accessibile al programma, per segnalare l'errore. Vedremo più avanti come analizzare la parola di stato. Quando in lettura si incontra la segnalazione di fine file, anche questa notizia viene riportata opportunamente nella parola di stato. La procedura descritta ha luogo sia per le operazioni INPUT # che per le operazioni GET #.

Per i dispositivi IEEE-488 avviene qualcosa di analogo, solo che il carattere richiesto deve arrivare entro 65 millisecondi; se questo non avviene si ha una segnalazione di FUORI-TEMPO (TIME OUT) nella parola di stato e quindi errore di lettura. Quando l'operazione di lettura è terminata viene liberato il BUS. Sebbene i dati vengano trasferiti al sistema operativo un carattere alla volta, il BASIC, per poterli manipolare, li ammucchia in un buffer di 80 posizioni. Per questa ragione è opportuno che i files non abbiano mai records più lunghi di 80 caratteri e che dopo 80 caratteri ci sia un RETURN. Se i records sono più lunghi, il sistema può essere disturbato e si possono avere errori che costringono a spegnere il PET per poi riaccenderlo e continuare dopo aver corretto opportunamente il programma. Se è necessario avere records più lunghi di 80 caratteri, si possono gestire con l'istruzione GET che non dà luogo ad inconvenienti. Comunque un settore disco non può contenere più di 254 caratteri.

6.10. PRINT # Uscita dei dati

Il formato dell'istruzione è:

PRINT # LF, lista di dati o variabili

dove LF è il numero logico di file usato nella OPEN e lista di dati o variabili, sono i dati da scrivere. I dati vengono trasferiti un carattere alla volta al dispositivo associato al file nella OPEN. Parecchi delimitatori del file, come ad esempio le virgole, vengono automaticamente cancellati dal BASIC. Se si desidera mantenere la virgola come separatrice di campi, si deve forzarne la scrittura o usando la funzione CHR$(44) oppure usando la stringa ",". Infatti la frase PRINT #6,A$,B$,C$ scrive sul file logico 6 le tre stringhe come una stringa unica, mentre la frase: PRINT#6,A$,",",B$,",",C$,","

scrive le tre stringhe con il separatore virgola e quindi poi potranno venire lette come 3 variabili. Lo stesso risultato si ottiene con la frase:

PRINT#6,A$,CHR$(44),B$,CHR$(44),C$.

Un'altra cosa da tener presente è che il BASIC formatta sempre l'uscita verso qualsiasi dispositivo come se stesse scrivendo sullo schermo. Per questa ragione le tre frasi PRINT appena scritte lasciano alcuni caratteri bianchi a completamento dei campi, questo perchè si sono separati i campi con la virgola. Se i campi si separano con il punto e virgola, questo non succede, cioè se scriviamo:

PRINT#6,A$;",";B$;",";C$

se A$ è di 5 caratteri, B$ di 8 caratteri e C$ di 7 caratteri, il record in uscita è così:

AAAAA,BBBBBBBB,CCCCCCC,

senza spazi bianchi a completamento a 10 dei campi. Con la PRINT precedente, che aveva la virgola come separatrice, l'uscita sarebbe stata:
AAAAA , BBBBBBBB , CCCCCCC.

Bisogna ricordare che, mentre per il video il ? equivale a PRINT, questo non è vero per PRINT#.

Esempio: Stampa dei primi 50 numeri naturali sulla stampante.

10 OPEN 5,4,0,	apre il file logico 5 sulla stampante 4 con SA=0 per semplice stampa
20 FOR K = 1 TO 50	stampa con un ciclo i 50 numeri andando a capo dopo ogni numero
30 PRINT # 5,K	
40 NEXT K	
50 CLOSE 5	chiude il file logico 5

Se volessimo stampare i 50 numeri senza andare a capo ogni volta, basta modificare la 30 così: 30 PRINT # 5,K; infatti il punto e virgola finale non fa andare a capo se non è fisicamente terminata la riga. Però in questo caso si deve aggiungere una frase 45 PRINT # 5 senza variabili per andare a capo terminati i numeri.

Se si vuole registrare su nastro la stessa sequenza di 50 numeri, basta cambiare nel programma di cui sopra la OPEN per quanto riguarda D=1 invece di 4 ed SA=1 oppure 2 a seconda che si voglia la scrittura senza registrazione di END OF TAPE o con tale registrazione. Nel caso del nastro i dati in uscita vengono caricati in un buffer di 192 caratteri uno alla volta e, quando il buffer è pieno, viene momentaneamente sospeso il caricamento dei dati, scritto il buffer sul nastro, e poi ripreso il caricamento dei dati.

6.11. Considerazioni sull'operazione di scrittura - Il comando CMD

Il comando PRINT # fa sì che il BASIC chiami una routine che inizializza il dispositivo IEEE-488 per l'uscita dei dati. La prima cosa che succede è che la normale uscita dei dati sul video viene invece diretta alla nuova periferica scelta con la OPEN. Allora viene inviato un segnale di ATTENTION al dispositivo tramite il bus e questo resta in uno stato di attesa. Il BASIC servendosi di un'altra routine invia un carattere per volta al dispositivo ricevitore. Quando l'operazione è terminata, il BASIC usando un'altra routine del sistema, stacca il dispositivo dal BUS e fa ritornare il video nello stato di primo destinatario dell'output. Quindi tra una frase di PRINT # e la seguente il BUS è libero.

Se si desidera avere più di un dispositivo attaccato al BUS e quindi colloquiare contemporaneamente con più di un dispositivo, si può usare il comando CMD. L'istruzione CMD è virtualmente identica alla PRINT #, solamente che, alla fine del trasferimento dei dati, non viene chiamata la routine che libera il BUS ed il ricevitore resta appeso al bus. Il sistema operativo continua a considerare l'ultimo dispositivo che è stato individuato dal comando CMD come il dispositivo di uscita primario. In conseguenza i comandi LIST e PRINT vengono diretti a questo dispositivo invece che al video. Per staccare il dispositivo dal BUS è necessaria una istruzione PRINT #.

Esempio: Per listare un programma sulla stampante:

10 OPEN 3,4	apre il file 3 su stampante per scrivere semplicemente
20 CMD 3	fa diventare la stampante il dispositivo primario di uscita
30 LIST	fa listare sulla stampante
40 PRINT # 3	fa staccare il BUS
50 CLOSE 3	fa chiudere il file.

6.12. Chiusura dei files

È opportuno che qualsiasi file logico aperto durante un programma sia chiuso prima della fine del programma. Per i files su nastro e su disco la chiusura è necessaria. È bene tener presente che le operazioni del PET possono diventare incoerenti se il numero totale dei files aperti è maggiore di 10. Se un file su nastro non viene chiuso, non viene registrato alcun END OF FILE alla fine del file. Se poi il file viene letto, il PET non può riconoscere il punto di fine del file e può considerare appartenenti al file dati di una precedente registrazione. Il formato della frase è il seguente: CLOSE LF.

Per i nastri che sono stati aperti con SA = 1 al momento del CLOSE, viene registrato un carattere di FINE FILE, mentre per quelli aperti con SA = 2 viene registrato anche un carattere di FINE NASTRO. Per i file IEEE-4888 che hanno un nome, e quindi per i dischi, viene inviata al momento della CLOSE una speciale sequenza di comandi con un indirizzo secondario che è un OR del carattere esadecimale "EO" e del SA dato nella OPEN. Questo permette al dispositivo di controllo dei dischi di chiudere il file.

6.13. Analisi degli errori nelle operazioni di Input/Output

Il concetto generale del sistema operativo del PET è quello di permettere all'utente di operare in formato libero; ciò consente di leggere o scrivere su nastri o dischi e di scrivere su stampanti nella maniera più comoda per l'utente. Dato che l'input/output è totalmente in formato libero, è molto importante che il sistema operativo sia in grado di informare l'utente quando si verifica un errore di trasmissione o si incontra un segnale di fine dati. Per facilitare la rilevazione degli errori nelle operazioni di I/O, il PET usa una parola di stato. Esiste cioè un byte, accessibile con il nome simbolico ST, che viene manipolato da ciascuna operazione di I/O e che i programmatore può richiamare a programma con il nome ST per analizzarne il contenuto e riconoscere l'errore verificatosi. Ciascuno degli 8 bits del byte ST ha un significato generale per tutte le operazioni ed un significato particolare per ciascun dispositivo di I/O, come possiamo vedere dalla tabella che segue.

Posizione bit in ST	**Valore numerico ST**	**Cassette magnetiche lettura**	**Dispositivi IEEE-488**	**Verifica + Load per cassette**
0	1		TIME OUT in scrittura	
1	2		TIME OUT in lettura	
2	4	Blocco corto		Blocco corto
3	8	Blocco lungo		Blocco lungo
4	16	Errore fatale		Qualunque
5	32	Errore nel Checksum		Errore nel Checksum
6	64	FINE FILE	END o IDENTIFY	
7	–128	FINE NASTRO	Dispositivo non presente	FINE NASTRO

Osservando la tabella precedente, si vede che per i dispositivi IEEE-488 si possono avere 3 tipi di errori:

1. il BUS non risponde ad una richiesta di attenzione, allora la routine del sistema alza il bit 7 di ST e stampa sul video DEVICE NOT PRESENT, oppure il BUS risponde correttamente ad una richiesta di attenzione, ma quando si tenta di scrivere il dispositivo non è presente fisicamente come risulta dal segnale basso NRFD o NDAC ed anche in questo caso si ha bit 7 di ST alzato e il messaggio al video.
2. Durante il trasferimento dei dati al dispositivo, il BUS non risponde entro giusto tempo o cessa di rispondere per mezzo di segnali NRFD e NDAC ambedue alti. In questo caso il bit 0 di ST viene alzato.
3. Durante la lettura tramite il BUS il dispositivo periferico non invia i segnale DAV entro 65 millisecondi, allora viene alzato il bit 1 di ST.

Per le cassette si analizzano gli errori solo in lettura; gli errori che possono essere rilevati sono:

1. Blocco corto (ST=4). Leggendo un blocco da nastro si incontra un segnale di spaziatura prima di aver letto i caratteri che ci si aspettava di trovare.
2. Blocco lungo (ST=8). Leggendo un blocco non si trova il segnale di spaziatura dopo aver letto il numero di caratteri che si aspettava.
3. Errore fatale (ST=16). Si sono riscontrati più di 31 errori nel primo blocco del blocco di controllo oppure si è trovato un errore che non può essere corretto perchè è presente in tutti e due i blocchi registrati (la registrazione su nastro è sempre doppia).
4. Errore di Checksum (ST=32). Durante la lettura o il LOAD di dati, viene calcolata una somma di controllo sui bits dei bytes letti e viene comparata con la somma di controllo registrata sul nastro al momento della scrittura. Le due somme non coincidono.
5. END OF FILES (ST=64). Viene incontrato il contrassegno di fine file.
6. END OF TAPE (ST=−128). Viene incontrato il contrassegno di fine nastro.

Come si è visto il PET non segnala mediante messaggi tutti gli errori che si possono verificare in I/O; per questo è necessario imparare a programmare usando la parola di stato ST. Bisogna però ricordare che ST cambia dopo ogni operazione di I/O e quindi va analizzata subito dopo l'operazione.

Esempio: Scrivendo

```
110 INPUT # 2, A
115 INPUT # 5, B
120 IF ST = 0 THEN 200
```

si controlla la parola di stato solo dopo la lettura dal file logico 5.

Scrivendo:

```
100 INPUT # 2, A
110 PRINT A
120 IF ST = 0 THEN 180
```

si controlla solo se la stampa è andata bene.

Un modo corretto di usare ST è questo:

100 INPUT # 2,A,B,C,	in 200 continua la normale elaborazione dopo la lettura
110 IF ST = 0 THEN 200	
120 IF ST = 64 THEN 300	in 300 continua l'elaborazione alla fine della lettura dei dati (fine-file) senza errori
130 IF ST = 2 THEN 400	in 400 si procede se il dispositivo IEEE-488 ha dato errore di time out.

Il controllo su ST si può anche fare sfruttando l'operazione logica AND e servendosi di una variabile M che faccia da maschera per il bit che in quel momento si desidera analizzare: IF ST AND M THEN ....

Quando si hanno collegati dei dispositivi lenti, come periferiche per campionamenti e simili (non le periferiche standard del sistema) si può sfruttare l'errore di TIME OUT per creare a programma un ciclo di attesa dell'input, uscendo dal ciclo quando il dato arriva e non si ha più TIME OUT.

6.14. L'uso della stampante

Abbiamo già visto che usando il comando CMD si può trasferire l'uscita dei dati dallo schermo alla stampante. Riepiloghiamo questo con degli esempi.

OPEN 5,4	apre il file logico 5 sulla stampante per stampa normale SA=0
PRINT # 5,"BUONA GIORNATA"	stampa BUONA GIORNATA sulla stampante
CLOSE 5	chiude il file logico 5
OPEN 5,4	apre il file logico 5 sulla stampante
CMD 5, "BUONA GIORNATA"	stampa BUONA GIORNATA sulla stampante e lascia la stampante aperta in attesa
PRINT # 5	stampa a vuoto, ma stacca la stampante
CLOSE 5	chiude il file logico 5

Il comando di PRINT a vuoto è necessario per staccare la stampante prima di chiudere il file.

OPEN 5,4	apre il file logico 5 sulla stampante
CMD 5,"BUONA GIORNATA UNO"	apre la stampante, stampa BUONA GIORNATA UNO e lascia la stampante aperta
PRINT # 5,"BUONA GIORNATA DUE"	stampa BUONA GIORNATA DUE e stacca la stampante
CLOSE 5	chiude la stampante
OPEN 5,4	apre il file logico 5
PRINT # 5,"PRIMA STAMPA"	stampa PRIMA STAMPA, ma la stampante non resta aperta
CMD 5,"SECONDA STAMPA"	stampa SECONDA STAMPA ma la stampante resta aperta
PRINT # 5: CLOSE 5	usa i due-punti per scrivere più istruzioni sulla stessa linea, con PRINT stacca la stampante e con CLOSE la chiude.
OPEN 5,4	apre il file logico 5
CMD 5	mette la stampante in attesa dell'output che prima usciva sullo schermo
LIST	fa listare quello che c'è in memoria sulla stampante (analogo effetto con comandi di PRINT per il video)
PRINT # 5	stampa a vuoto, ma stacca la stampante
CLOSE 5	chiude il file logico 5

Questi esempi illustrano l'uso della stampante in modo diretto: è necessario fare molte prove sul PET per impadronirsi bene di questi concetti prima

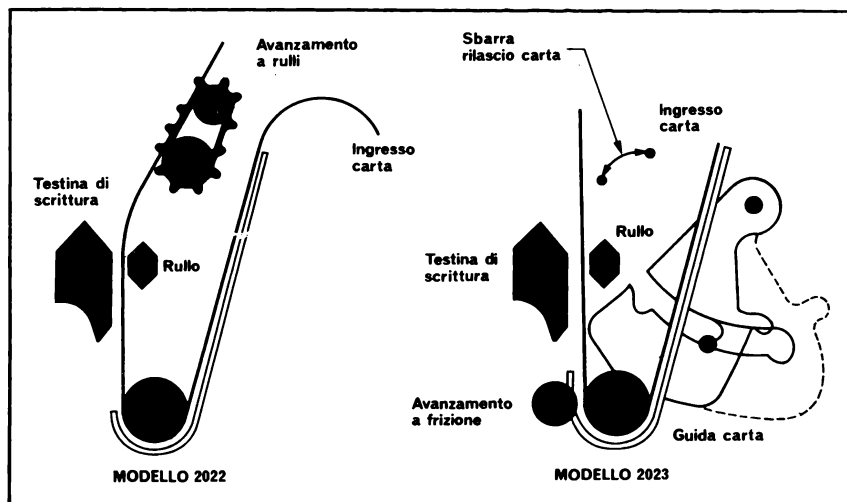



Figura 6-4. Scorrimento della carta nelle stampanti 2022 e 2023.

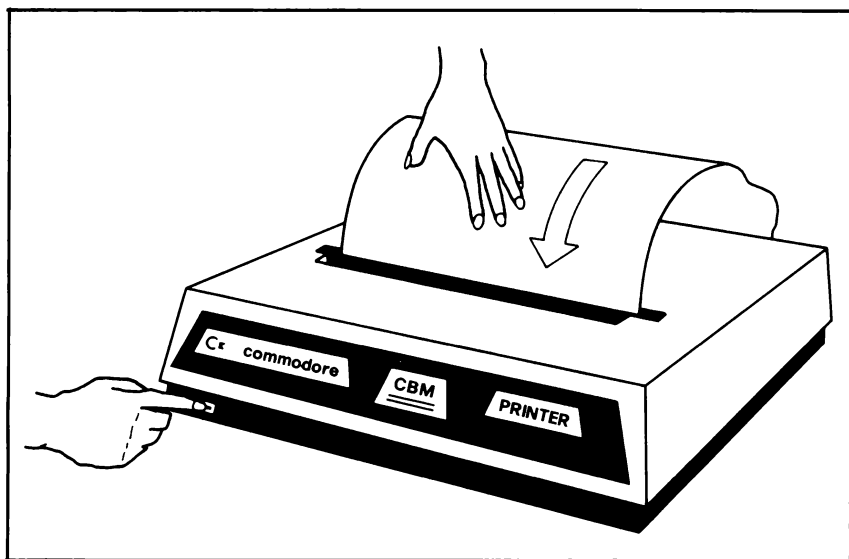



Figura 6-5. Come inserire la carta nella stampante 2023.

di passare all'uso della stampante sotto controllo del programma.

Esempio di uso della stampante sotto controllo del programma.
Scrivete NEW - RETURN e poi il programma che segue:

```
10 OPEN 3,4
20 CMD 3
30 PRINT "QUESTO PROGRAMMA ILLUSTRA UNA PROVA
   DELLA STAMPANTE"
40 LIST
```

poi date RUN avendo naturalmente acceso la stampante. Vedrete comparire la scritta QUESTO PROGRAMMA ILLUSTRA UNA PROVA DELLA STAMPANTE e poi per effetto di LIST la lista del programma. Dato che l'esecuzione della frase LIST fa uscire dal programma, poi date in modo diretto i due comandi:

```
PRINT # 3
CLOSE 3.
```

Fate ora questa altra prova, nella quale userete delle speciali funzioni di stringa il cui studio verrà fatto più avanti.

Sempre dopo NEW scrivete questo programma:

```
10 OPEN 4,4
20 PRINT # 4, "H"; CHR$(1)"E", CHR$(1)"L"; CHR$(1)"L";
   CHR$"O"
30 CLOSE 4
```

dopo date RUN otterrete la scritta HELLO con le lettere che via via ingrandiscono per effetto della funzione CHR$(1).

Provate ora quest'altro programma, nel quale si fa uso dei caratteri OFF/RVS e CRSR DOWN e per effetto del quale si ha la stampa di una intestazione e di tutti i caratteri del PET.

Dopo il solito NEW, scrivete questo programma:

```
10 OPEN 4,4
20 FOR I=32 TO 95 :A$=A$+CHR$(I):NEXT
30 FOR I=160 TO 223 :B$=B$+CHR$(I):NEXT
40 C$="▪"+A$
50 D$="▪"+B$
60 E$="▪"+A$
70 F$="▪"+B$
80 G$="▪"+C$
90 H$="▪"+D$
100 PRINT#4,CHR$(1)" PET CARATTERI DELLA STAMPANTE "
110 PRINT#4:PRINT#4:PRINT#4
120 PRINT#4,A$
130 PRINT#4,B$
140 PRINT#4,C$
150 PRINT#4,D$
160 PRINT#4,E$
170 PRINT#4,F$
180 PRINT#4,G$
190 PRINT#4,H$
200 CMD 4
210 PRINT:PRINT:PRINT:PRINT:PRINT
220 LIST
```

ora date RUN e vedrete stampati tutti i caratteri del PET in campo normale e inverso, in minuscola e maiuscola.

```
PET CARATTERI DELLA STAMPANTE

 !"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]↑←
 !"#$%&'()*+,-./0123456789:;<=>?@abcdefghijklmnopqrstuvwxyz
ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]↑←
```

Dopo date al solito

PRINT # 4
CLOSE 4 in modo diretto per chiudere correttamente.

6.15. I diversi formati di stampa

L'indirizzo secondario SA per la stampante può assumere valori da 0 a 6 permettendo una vasta gamma di tipi di stampa. SA = 0 è il modo più semplice di stampare e cioè esattamente come sono i dati, 80 caratteri per riga, (o più a seconda del tipo di stampante) se dopo 80 caratteri non si incontra il RETURN, la stampante va a capo automaticamente e continua. SA = 1 è il codice che fa stampare secondo un formato precedentemente predisposto, usando il codice*SA=2*. Se non si è predeterminato il formato mediante SA=2, il codice SA=1 fa stampare come SA=0. Per usare bene questo formato si deve procedere così:

- aprire un file logico sulla stampante con SA=2
- aprire un altro file logico sulla stampante con SA=1 (i numeri dei file logici devono essere diversi),
- dare un PRINT # con la stringa che dà il formato per il primo file logico,
- dare un PRINT # con i dati da stampare per il secondo file logico ed i dati del secondo PRINT sono stampati con il formato predisposto dal primo.

Per esempio:

10 OPEN 2,4,2	LF=2	SA=2
20 OPEN 1,4,1	LF=1	SA=1
30 PRINT # 2,"$$$.99"	maschera di stampa per LF=2	
40 PRINT # 0.05	stampa il numero decimale 0,05 secondo il	
50 CLOSE 2:CLOSE 1	formato di stampa prima predisposto	
	chiude i files.	

Dando il RUN si ottiene $.05 in stampa. Al momento dell'esecuzione della frase 30 il formato di stampa viene memorizzato e conservato per la stampa fino a quando non ne viene dato un altro. A questo punto è necessario studiare come si devono formare le stringhe per preparare le stampe. Chiamiamo queste stringhe "Maschere di stampa". I caratteri usabili nelle maschere sono:

- Caratteri numerici: 9 Z $ S —
- Caratteri alfanumerici: A
- Carattere di salto: il blank o spazio

Vediamo il significato di ogni carattere:

9 : specifica una cifra numerica, se al momento della stampa la cifra manca, viene sostituita da uno spazio.

Z : specifica una cifra numerica, se la cifra manca al momento della stampa, viene forzato al suo posto uno zero. Serve se si desidera avere zeri di riempimento in un campo.

$: se si ha un solo $ il campo viene stampato preceduto da $, se si hanno più $ ripetuti si ha la stampa del campo allineato a destra e con un solo $ prima della cifra più significativa.

S : il campo numerico che viene stampato è preceduto dal segno + o — in posizione fissa.

: definice la posizione del punto decimale e viene stampato dove si è messo nella maschera.

— : viene stampato un meno dopo il numero se questo è negativo, mentre se è positivo non viene stampato il segno.

Riportiamo di seguito alcuni esempi di maschere di stampa o, come si usa anche dire, di formati di stampa.

Maschera o formato	**Dati da stampare**	**Risultato in stampa**	**Commenti**
AAAAA	ABC	ABC	stringa orientata a sinistra
AAAAA	ABCDEFG	ABCDE	come prima ma con troncamento di 2 caratteri
$$$$$	99	$99	numeri allineati a destra
$9999	99	$99	$ è rimasto fisso
$99.99	77	$77.00	
$99.99	—77	$77.00	segno perso perchè non previsto
$99.99—	—77	$77.00—	
$99.99—	77	$77.00	
S$99.99	77	+$77.00	
S$99.99	—77	—$77.00	
ZZ.999	77	77.000	
ZZ.Z99	77	077.00	
999.99	77	77.00	
.99	77	.**	compaiono gli asterischi per maschera insufficiente
.99	.001	.00	
S.999	1.5E—02	+.015	
Z.999—	1.5E—02	0.015	
Z.999—	—1.5E—+02	0.015—	

La stampante può trattare fino a 10 cifre significative con un esponente per i numeri in notazione esponenziale da − 99 a + 99. Per stampare le stringhe si deve usare la funzione CHR$(29), carattere di skip, per segnalare la fine di una stringa da stampare in un campo. All'interno del campo gli spazi che precedono sono troncati, il campo è appoggiato a sinistra e riempito a destra da spazi. Il blank shiftato, CHR$(160) non viene cancellato se è messo nella prima posizione del campo.

Esempio:

OPEN 1,4,2
OPEN 2,4,1

dando sempre queste due OPEN le PRINT che seguono lavorano come descritto.

PRINT # 1,"AAA   AAA   AAA"
PRINT # 2,"ABC" CHR$(29) CHR$(160) CHR$(29) "DEF"
dà luogo a ABC   DEF
PRINT # 1,"A AA   AAA"
PRINT # 2,"CBM"CHR$(29)"CBM"CHR$(29)"CBM"
dà luogo a C   CB   CBM
PRINT # 1,"AAA AAA   AAAA"
PRINT # 2,"PET"CHR$(29)"PET"CHR$(29)"PET"
dà luogo a PET PET   PET

Nelle maschere di stampa si possono usare dei "letterali" cioè dei caratteri che si vuole vengano stampati come sono e non usati per formattare altri dati. Per ottenere ciò bisogna far precedere i letterali dal simbolo di REV OFF.

PRINT # 1"—\AAAA—\" dove — è REV OFF
e \ è la sbaretta verticale
PRINT # 2 "ABC" dà luogo a: \ABC\

Continuiamo lo studio dei diversi modi di stampa precisando il significato degli altri valori possibili per SA. *SA=3* serve per forzare il numero di linee per pagina ad un valore diverso da 66 che è lo standard. Per ottenere ciò si deve aprire un file logico per la stampante con SA=3 e dare una PRINT per quel file con il numero di linee volute. Il numero di file logico serve solo per questa operazione e deve essere diverso dai numeri usati per le altre operazioni di stampa. Esempio:

OPEN 8,4,3
PRINT # 8,20

fa stampare 20 linee per pagina. Perchè si abbia il controllo del numero di linee per pagina si deve attivare questa funzione usando il carattere di

PAGING ON che è CHR$(147) all'inizio del programma. Se si vuole disattivare il controllo del numero di linee per pagina, si deve usare il carattere PAGING OFF che è CHR$(19). Quando si fanno 66 linee per pagina in 66 è compreso il conto delle linee bianche all'inizio ed alla fine della pagina.

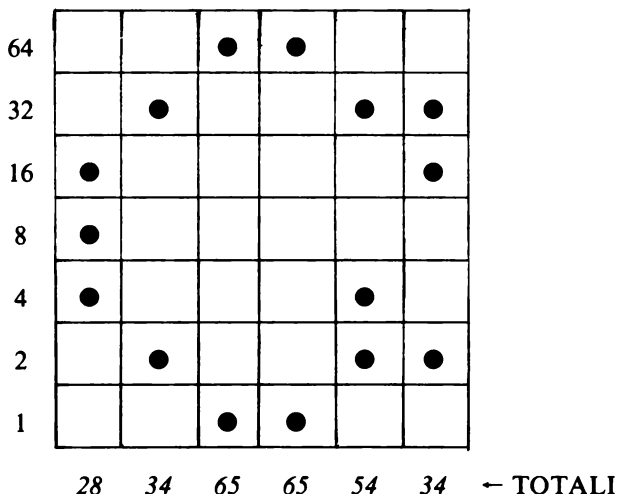
SA = 4 serve per ottenere un messaggio diagnostico alla stampante in caso di errore. Se la maschera è errata e si è attivato il controllo con SA=4 si ha la stampa della maschera con un puntatore all'errore così:

```
99 01
ZZ  ZZ  XZ  ZZ
         ↑
**** BAD FORMAT
```

Per altri errori si possono avere i messaggi seguenti:

```
****           BAD FORMAT             ****
****     DATA - FORMAT MISMATCH       ****
****          BAD COMMAND             ****
****     EXCESS LINES PER PAGE        ****
```

Se non si è attivato questo controllo SA= 4 i dati errati vengono mandati alla linea di stampa, ma i dati vengono stampati come se si avesse SA=0. In tutti i casi, anche senza SA=4, se si ha supero di capacità per un campo numerico, tale campo viene riempito di asterischi e si passa alla stampa con SA=0.

SA=5 permette di definire un carattere programmabile. Questa funzione vi permette di creare un vostro carattere grafico da stampare. Facciamo un

esempio riferendoci al marchio della Commodore Business Machines. Si procede così: si deve disegnare una matrice (tabella) formata da 7 righe per 6 colonne divisa quindi in 6 x 7 = 42 quadratini. In questa matrice si deve disegnare il carattere voluto mettendo un punto nei quadratini interessati. Ogni punto rappresenta un bit il cui valore è segnato a sinistra della matrice. Sotto la matrice scrivete per ogni colonna il totale dei valori dei bits della colonna. Questi totali devono entrare con una frase DATA come costanti nel vostro programma. Il programma servendosi dei valori del DATA crea una stringa con la funzione CHR$ e la trasmette alla stampante usando SA=5. Per passare dalle maiuscole alle minuscole si usa il carattere CRSR UP e CRSR DOWN. Studiate con cura il programma che segue.

I valori da caricare con la frase DATA sono in ordine: 28,34,65,65,54,34

```
10 DATA 28,34,65,65,54,34
20 OPEN 5,4,5
30 FOR I=1 TO 6:READ A:A$=A$+CHR$(A):NEXT
40 PRINT#5,A$
50 OPEN 4,4
60 FOR I=1 TO 10
70 PRINT#4,CHR$(1)CHR$(254)" COMMODORE BUSINESS MACHINES"
80 NEXT
90 CLOSE 5
100 CLOSE 4
```

```
C Commodore Business Machines
C Commodore Business Machines
C Commodore Business Machines
C Commodore Business Machines
C Commodore Business Machines
C Commodore Business Machines
C Commodore Business Machines
C Commodore Business Machines
C Commodore Business Machines
C Commodore Business Machines
```

Commenti: L'istruzione 20 attiva LF=5 con SA=5 per il marchio e tale file va chiuso a fine programma. La 50 apre la stampante come file 4 per stampa con SA=0. La 30 con un ciclo percorso 6 volte prepara la stringa A$ con la funzione CHR$(A), tale stringa serve per il marchio. La 40 memorizza la stringa A$ nella memoria della stampante. Con il ciclo eseguito dalle frasi da 60 a 80 viene stampato 10 volte il marchio e la scritta. La funzione CHR$(1) allarga i caratteri, mentre la funzione CHR$(254) scrive il carattere programmato che si trova memorizzato nella memoria della stampante. Le frasi 90 e 100 chiudono i files.

SA=6 vale solo per la stampante 2022 e consente di modificare la distanza tra le linee di stampa. Lo standard è di 6 linee per pollice. Si procede così:

```
OPEN 6,4,6
PRINT # 6,CHR$(18)
```

la stampante si comporta così: stampa per ogni pollice un numero di linee pari a quel numero che moltiplicato per 18 dà 144 e quindi, in questo caso 8. Infatti 18 x 8 = 144. Tenendo conto di questo, si vede che lo standard di 6 linee per pollice corrisponde a PRINT # 6,CHR$(24) infatti 24 x 6 = 144. Se si scrive PRINT # 6,CHR$(144) si ottiene 1 linea per pollice.

Nella tabella che segue riepiloghiamo i caratteri di controllo speciali che si possono usare per modificare il modo di stampare all'interno di una linea di stampa. Qui diamo un breve commento.

Allargamento caratteri. Si ottiene con la funzione CHR$(1), il carattere invece di essere contenuto in una matrice di 7 righe e 6 colonne, è contenuto in una matrice di 7 righe e 12 colonne. Usando più volte di seguito la funzione, si ottiene un maggiore allargamento. Usando la funzione inversa CHR$(129) si ritorna al carattere normale.

Tabella dei caratteri di controllo speciali

Funzione	Codice	Valore ASCII	Tastiera
Allargamento	CHR$(1)	SOH	
Non allargamento	CHR$(129)		
Paginazione on/reset	CHR$(147)		SHIFT & CLR/ HOME
Paginazione off	CHR$(19)	DC3	CLR/HOME
RVS ON	CHR$(18)	DC2	OFF/RVS
RVS OFF	CHR$(146)		SHIFT & OFF/ RVS
Ritorno carrello	CHR$(13)	CR	RETURN
Ritorno carrello senza spazio	CHR$(141)		
Spaziatura a capo	CHR$(10)	LF	
Maiuscole	CHR$(145)		SHIFT & CRSR
Minuscole	CHR$(17)		CRSR
Spaziatura orizzontale	CHR$(29)		CRSR RIGHT
Virgolette	CHR$(34)	LF	CRSR RIGHT
Va a nuova pagina	CHR$(12)		

Nota: Raccomandiamo molto esercizio sul Pet per imparare a stampare correttamente.

Paginazione. Si ottiene usando la funzione CHR$(147), altrimenti la stampante scrive con continuità. La funzione di paginazione fa scrivere 66 righe per pagina comprendendo 3 righe bianche all'inizio ed alla fine. Il numero di linee per pagina può essere alterato usando l'indirizzo secondario SA=3. Quando si lavora con paginazione, per cambiare foglio si deve usare la funzione CHR$(19).

Maiuscole e minuscole. Contrariamente che sul video si possono ottenere righe miste usando le funzioni CHR$(145) e CHR$(17).

Campo inverso. Si ottiene usando la funzione CHR$(18) che modifica la matrice facendo apparire bianco su verde. Non usare mai questa possibilità per più di 5 linee consecutive per non danneggiare la testina scrivente. La funzione CHR$(146) fa ritornare al modo normale.

Ritorno carrello senza spaziatura. Si ottiene con la funzione CHR$(141) e permette di sovrastampare sulla stessa linea. Se si usa il ritorno carrello, vengono annullate le funzioni di: campo inverso, allargamento caratteri, virgolette.

Virgolette. Se si trasmettono un numero dispari di virgolette diventano visibili i caratteri di controllo.

CAPITOLO 7

L'USO DEI FLOPPY DISK

7.1. Comandi BASIC fondamentali per i dischi (unità 3040)

I dischi comunicano con il PET tramite 16 canali adibiti a scopi diversi. Per il momento consideriamo solo il canale 15 che serve per i comandi e per i messaggi di errore. In seguito vedremo l'uso degli altri canali. Ricordando che la periferica disco ha D=8, *l'istruzione di apertura* è:

OPEN LF,D,SA
e precisamente:

OPEN LF,8,15	apre il file logico LF su disco, *per inviare comandi al disco.*

L'istruzione di chiusura è come già visto:

CLOSE LF

Per *trasferire un programma dalla memoria al disco* si scrive:

SAVE "dr:fn",8	dove dr = 0 per l'unità 0 e dr = 1 per l'unità 1 fn = nome del file programma da memorizzare e deve essere al massimo di 16 caratteri

Per *verificare un programma appena scritto su disco* con il contenuto della memoria, si usa:

VERIFY "dr:fn",8	dove dr ed fn hanno il significato visto sopra.

La verifica avviene byte contro byte. Se il confronto non dà buon risultato, si ha un messaggio di errore. Per verificare un programma per il quale si è appena usato il SAVE, si può anche scrivere:

VERIFY "*",8.

Per *ricopiare programmi dal disco in memoria* si scrive:

LOAD "dr:fn",8

Esiste un comando che *trasmette al disco una stringa-comando;* esso è:
PRINT # LF, "stringa-comando"
dove LF è il numero del file e la stringa-comando può contenere i seguenti comandi:

- N (NEW) per predisporre i settori sul disco e inizializzarlo,

- I (INIZIALIZE) per allineare la testina di lettura, leggere la directory e caricare nel DISK OPERATING SYSTEM i dati necessari per gestire il disco,
- V (VALIDATE) per creare la BAM (Block Availability Map) considerando i dati validi e per inizializzare il disco,
- D (DUPLICATE) per duplicare un disco,
- C (COPY) per copiare e/o fondere files tra due dischi diversi o sullo stesso disco,
- R (RENAME) per cambiare nome ad un file,
- S (SCRATCH) per cancellare un file.

La directory è l'indice del contenuto del disco e viene scritta dal sistema nella traccia 18, che pertanto non può essere usata dall'utente. Per i dischi la directory può contenere al massimo 152 registrazioni, quindi non si possono avere più di 152 file, tra programmi e file sequenziali.

7.2. Preparazione di un disco nuovo per l'uso

Per poter usare un disco bisogna predisporre la divisione in settori del disco stesso. Ricordatevi di non dare o togliere corrente all'unità dico con un disco inserito.

Procedura:

OPEN 1,8,15	apre il file logico 1 sul dispositivo 8 ed usa il canale 15
PRINT # 1,"Ndr: nome-disco,id"	dove dr=0 per unità zero e dr=1 per unità 1; nome-disco è il nome che si vuole assegnare al disco al massimo di 16 caratteri; id è un identificatore del disco di 2 caratteri e si può anche omettere.
CLOSE 1	chiude il file logico 1

Con queste 3 operazioni avete scritto in ognuna delle 35 tracce del disco gli identificatori dei settori, azzerato l'indice delle registrazioni sul disco e riinizializzato la BAM (Block Availability Map).

7.3. Modalità di inserimento dei dischi nell'unità a dischi

Ripetiamo quanto detto nel paragrafo precedente:

NON TOGLIERE MAI CORRENTE ALL'UNITA' A DISCHI CON I DISCHI DENTRO ED ANALOGAMENTE NON DARE MAI CORRENTE ALL'UNITA' A DISCHI CON I DISCHI GIA' DENTRO.

Altra cosa da tener presente è la seguente: la finestrella dell'alloggiamento dei dischi non va chiusa fino a quando non si vede accendere la spia rossa

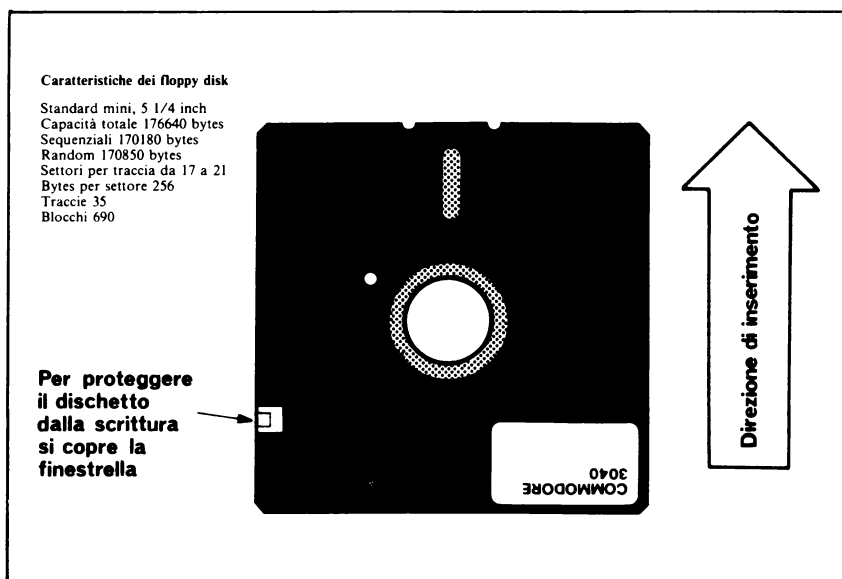



Figura 7-1. Floppy disk

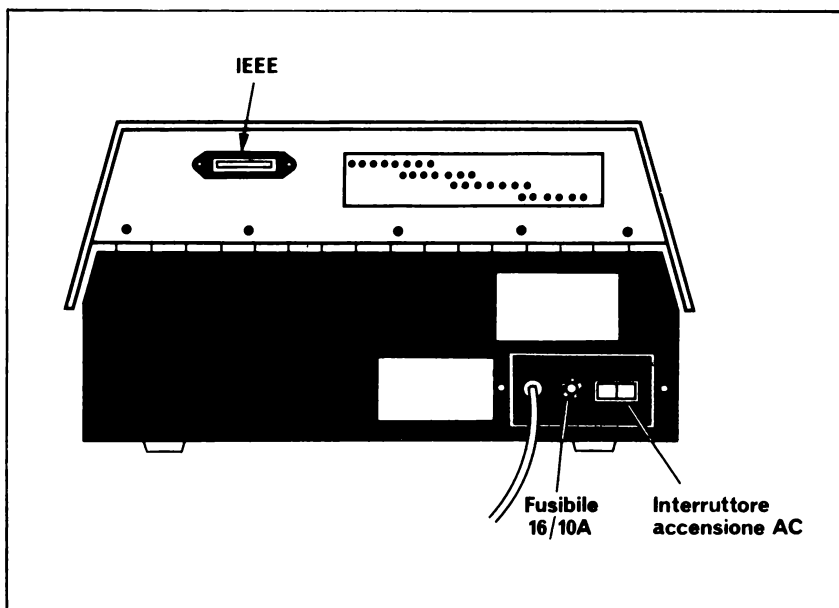



Figura 7-2. Parte posteriore dell'unità 3040.

del disco e non si sente il rumore del motore; a quel punto deve essere chiusa. Con la finestrella aperta il disco si sistema meglio nel dispositivo in fase di partenza. Se si vuole chiudere subito la finestrella, si può farlo, dopo averla fatta bilanciare un po' delicatamente, sempre per permettere al disco di inserirsi bene nell'alloggiamento. Controllate dopo le operazioni disco la spia rossa centrale; se resta accesa significa che si è avuto un errore.

7.4. Preparazione di un disco già usato per un nuovo uso

Se si svolge la procedura del paragrafo 7.2. il disco viene riformattato ed azzerato. Se si desidera solo azzerarlo, ed eventualmente modificargli il nome, ma non i due caratteri di identificazione, si può usare la seguente procedura, che è più veloce:

```
OPEN 1,8,15
PRINT # 1,"Ndr:nome-disco"
CLOSE 1
```

cioè non si mette nella stringa comando la virgola dopo il nome del disco, seguita dai due caratteri di identificazione. In tale modo la id del disco resta immutata, ma viene azzerato l'indice del disco e di conseguenza cancellati tutti i files precedentemente registrati.

7.5. Come si usa un disco già inzializzato o già anche parzialmente scritto

L'operazione di inizializzazione vista nei precedenti paragrafi, oltre a preparare il disco per l'uso, ha anche l'effetto di allineare le testine di lettura nella posizione corretta per poter lavorare con i dischi. Quando si monta un disco già in uso si deve fare l'operazione di allineamento della testina in questo modo:

```
OPEN 1,8,15
PRINT # 1,"Idr"
CLOSE 1
```

dove dr=0 per disco 0 e dr=1 per disco 1; se si vogliono allineare i due dischi, basta scrivere "I" senza dr. Chiaramente i comandi che noi usiamo in modo abbreviato, possono essere usati completi come elencati nel paragrafo 7.1. Analogamente ricordiamo che in tutti questi esempi si usa sempre come numero logico del file il numero 1, ma si potrebbe usare anche altro numero. Invece 8 e 15 devono restare tali; 8 è il numero del dispositivo e 15 è il numero del canale per i comandi.

7.6. Scrittura di un programma su disco

Supponendo di avere un programma nella memoria del calcolatore per memorizzarlo su disco, si opera così:

SAVE "dr:nome-programma",8	scrive il programma sul disco dr e gli assegna il nome nome-programma
VERIFY "dr:nome-programma",8	verifica che il programma sia stato scritto bene; se no segnala errore.

dr=0 per disco 0 e dr=1 per disco 1.

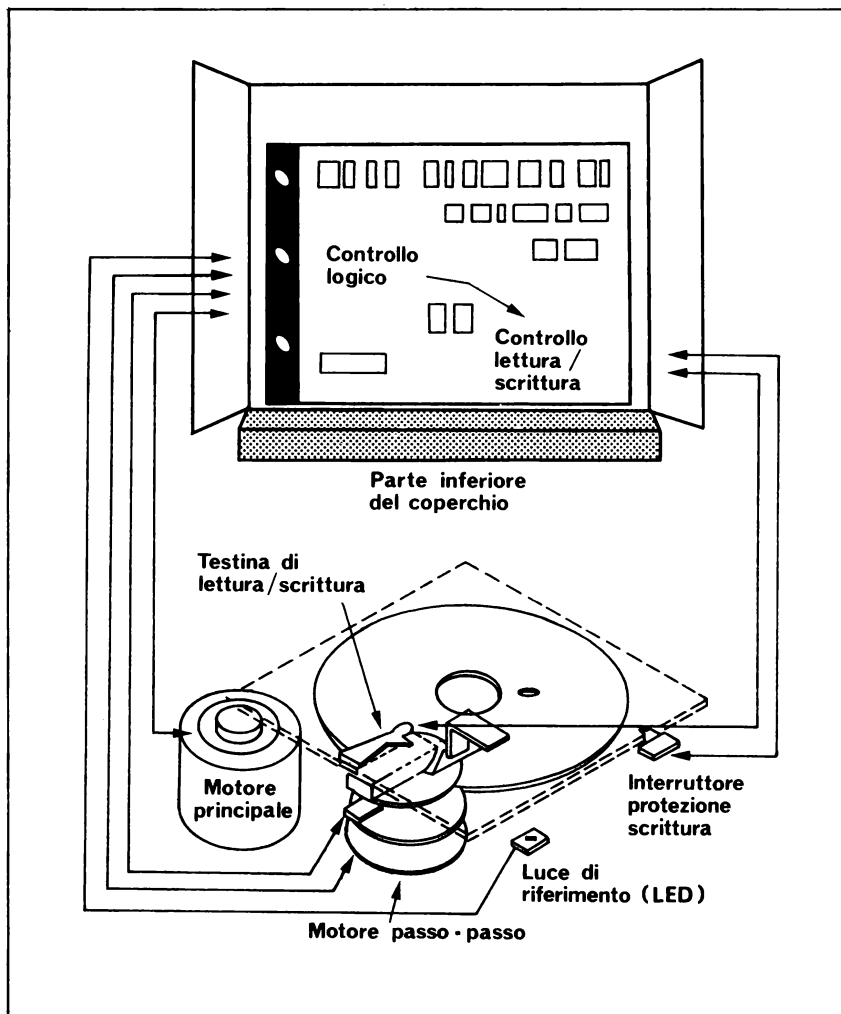



Figura 7-3. Unità 2040 - Collegamenti interni.

7.7. Lettura di un programma da disco in memoria

Per caricare un programma da disco in memoria, si procede così:

LOAD "dr:nome-programma",8

alla fine del caricamento si vede sul video READY.

7.8. Lettura della DIRECTORY (INDICE) del disco

In ogni disco esiste un indice dei contenuti che si chiama DIRECTORY. Per caricare in memoria tale indice si procede così:

LOAD "$dr",8 — dove il simbolo dollaro ($) significa che si vuole la directory e dr è il numero del disco interessato all'operazione.

Con questo comando la Directory va in memoria e si può listarla o sul video o sulla stampante se si omette dr viene letta la DIRECTORY dei due dischi.

7.9. Per sistemare un disco già in uso

Su un disco in uso si possono avere anche dati non validi, come file non corettamente chiusi. Con il comando VALIDATE si crea la BAM (Block Availability Map), cioè la mappa dei blocchi disponibili cancellando anche i file non in stato buono. Naturalmente prima di dare questo comando, il

Figura 7-4. Unità a floppy disk del CBM.

disco deve essere stato inizializzao con INITIALIZE. Questo comando richiede la OPEN.

OPEN 1,8,15
PRINT # 1,"Vdr"   dr=0 per disco 0 e dr=1 per disco 1.

7.10. Duplicazione disco

Con questo comando si crea una copia identica di un disco, cioè si ottiene un disco con la *stessa id,* lo *stesso nome* e lo *stesso contenuto.* Si deve montare su una unità il disco da copiare che chiamiamo *Sorgente*, e sull'altra il disco su cui copiare che chiamiamo *Destinazione.* Il disco sorgente deve essere inizializzato per allineare la testina. Il disco destinazione deve essere inizializzato con I se è già stato usato o con N se è nuovo. Per duplicare si opera così:

OPEN 1,8,15
PRINT # 1, "Ddr-destinazione=dr-sorgente"

attenzione a non invertire; a sinistra dell'uguale il disco destinazione e a destra il sorgente.

Esempio: OPEN 1,8,15
PRINT # 1,"D0=1" copia sul disco 0 il disco 1
mentre: PRINT # 1,"D1=0 copia sul disco 1 il disco 0.

7.11. Copia di files

Il comando COPY consente di copiare files tra due dischi diversi o sullo stesso disco. Si può usare questo comando anche per concatenare files di dati. Naturalmente i dischetti devono essere propriamente inizializzati.

PRINT # LF,"Cdr-destinazione:nome-file=dr-sorgente:nome-file, dr-sorgente;nome-file,...."

Esempio:

PRINT # 1,"C1:PROGR1=0:PROGR1" copia sul disco 1 il programma PROG1 prendendolo dal disco 0

PRINT # 1,"C1:DATIL=0:DATIA,0:DATIB,0:DATIC" copia sul file DATIL nel disco 1 i 3 files DATIA, DATIB e DATIC del disco 0.

Il file sorgente deve essere stato aperto prima della copiatura:

OPEN 1,8,15, "dr: sorgente".

7.12. Cambiamento nome al file su disco

Per cambiare il nome ad un file già esistente sul disco si usa il comando che segue. Il nuovo nome non deve ancora essere presente sul disco.

PRINT # 1,"Rdr:nome-nuovo=nome-vecchio".

7.13. Come si cancellano i files

Il comando SCRATCH serve per cancellare i files da disco. Si possono cancellare anche più files contemporaneamente. Il comando è:

PRINT # 1, "Sdr:nome-file,dr:nome-file,..."

dr=0 per disco 0,dr=1 per disco 1, dr=: significa riferimento all'ultimo disco utilizzato, se dr manca si riferisce ad ambedue i dischi.

PRINT # 1,"S1:*"	cancella tutti i files del disco 1.
PRINT # 1,"S1:BAO*"	cancella tutti i files che hanno il nome che comincia per BAO.
PRINT # 1,"S1:?????.SCR"	azzera tutti i files che hanno nome composto da 5 caratteri e terminante con .SCR.

I file da cancellare devono essere stati aperti come in 7.11.

7.14. Trattamento dei files di dati su disco

Nei paragrafi precedenti abbiamo visto come si trattano i files che contengono programmi, cioè di tipo PRG, che vengono scritti su disco con SAVE e letti in memoria con LOAD. Abbiamo visto alcuni comandi per operazioni disco di utilità generale. Tutte le operazioni viste si trattano usando il canale 15. Nei prossimi paragrafi vedremo come si trattano i files di dati. Questi files di dati possono essere di tipo SEQUENTIAL o USER e per trattarli useremo i canali da 2 a 14.
Cominciamo a vedere la frase OPEN:

OPEN LF,D,SA,"DR:FN,FT,MODE"

dove:

LF = numero logico del file
D = dispositivo (numero)
SA = numero del canale da 2 a 14
DR = 0 o 1, numero del disco
FN = nome del file
FT = tipo del file: SEQ o USR o PRG; che si possono abbreviare in S,U,P.
MODE = READ per input e WRITE per output; abbreviati in R e W.

Esempi:

OPEN 2,8,2,"0:FILE 1, SEQ, WRITE" apre il file logico 2 su disco usando il canale 2; il disco è lo 0, il nome del file è FILE 1, il file è di tipo sequenziale e si vuole scrivere.

OPEN 3,8,9,"1:TEST DATA,PRG,WRITE" apre il file logico 3 su disco usando il canale 9, il disco è 1, il nome del programma TEST DATA, il tipo programma e si vuole scrivere.

OPEN 8,8,8,"0:NUM,USR,READ" apre il file logico 8 su disco usando il canale 8, il disco è 0, il nome del file NUM, il tipo USR per leggere.

Se si desidera *riscrivere un file* che esiste già, si scrive:

OPEN 3,8,5"@1:JDATI,USR,WRITE" apre il file logico 3 sul disco, usando il canale 5, il disco è sull'unità 1, il nome del fle da riscrivere è JDATA di tipo USR. Il simbolo @ prima del numero dell'unità disco significa riscrittura. Si può anche dare la stringa di specificazione del file e dell'operazione richiesta sotto forma di variabile stringa:

OPEN 1,8,14,FL$ dove FL$ = "DR:FN,FT,MODE"

con questo metodo si può, per esempio, leggere da tastiera la stringa FL$. Si può anche dare una stringa variabile solo in modo parziale:

OPEN 1,8,14,FL$+",SEQ,WRITE" allora FL$="DR:FN"

Per chiudere si deve usare la CLOSE scritta così:

CLOSE LF

cioè si chiude il filo logico, ma questa chiusura ha per effetto di chiudere il file logico ed il canale ad esso associato nella OPEN. Quando si chiude un file aperto per WRITE, viene scritto l'ultimo blocco su disco e poi chiuso. Bisogna are attenzione che quando si usano i comandi INITIALIZE, NEW, DUPLICATE o VALIDATE, vengono distrutte le precedenti aperture di canali per il disco in questione. Lo stesso capita se si usa LOAD. Inoltre bisogna tener presente che quando si chiude un canale comandi (il 15) vengono chiusi anche tutti i canali dati aperti. Vediamo questi esempi:

OPEN 1,8,15	apre il canale comandi 15 con file logico 1
OPEN 3,8,2,"0:FILE1,SEQ, WRITE"	apre i canali dati 2 e 5 per i dati
OPEN 4,8,5,"0:FILE2,SEQ, WRITE"	
PRINT # 3,"DATI IMPORTANTI"	scrive sui file logici 3 e 4
PRINT # 4,"ANCORA DATI"	
OPEN 3,4	viene aperto per errore il file logico 3 (già aperto) sulla stampante

Il sistema scrive:

?FILE OPEN ERROR?

Dato che si è avuto un errore, sono chiusi tutti i file logici del programma salvo i canali del disco. Per chiudere correttamente i canali dovete fare così:

OPEN 1,8,15

CLOSE 1

con questo CLOSE del file logico 1 collegato al canale comandi 15, vengono chiusi tutti gli altri canali.

SCRITTURA DEL FILE DISCO

Si usa il comando PRINT # citando il numero logico del file usato nella OPEN. La lista di variabili da scrivere deve terminare con: CHR$(13); cioè si deve dare come chiusura il carattere "RITORNO CARRELLO", che si ottiene con CHR$(13), e terminare con un punto e virgola. Se non si termina con un punto e virgola, il sistema manda su disco un ritorno carrello ed una spaziatura verticale (CARRIAGE RETURN-LINE FEED) e poi quando si rilegge si trovano questi caratteri. Esempio corretto:

PRINT # 2,"SONO CONTENTO";CHR$(13);

con questo comando si scrive sul file logico 2 precedentemente aperto.

LETTURA DEL FILE DA DISCO

Si possono usare i due comandi INPUT # e GET #. Esempio:

INPUT # LF, lista di variabili separate da virgola. Naturalmente bisogna che la lista di variabili concordi con i tipi di dati che si vogliono leggere da disco.

GET # LF, lista di variabili separate da virgola. Ricordiamo che la GET legge un carattere alla volta, cioè un carattere per ogni variabile della lista.

ERRORI DISCO

Quando state lavorando con i dischi, se si verifica un errore, si accende la spia rossa collocata tra le due unità. Per poter proseguire bisogna spegnere l'indicatore d'errore, questo si ottiene inserendo nei programmi una routine di errore la quale evidenzia al video l'errore accaduto e resetta (spegne) l'indicatore di errore. La routine che dovete scrivere, deve aprire il canale per i messaggi di errore, che è ancora il 15 e richiedere da una a quattro delle seguenti variabili nell'ordine che esponiamo:

A$ numero del messaggio di errore
B$ messaggio di errore
C$ 00 o il numero della traccia del disco
D$ 00 o il numero del settore del disco.

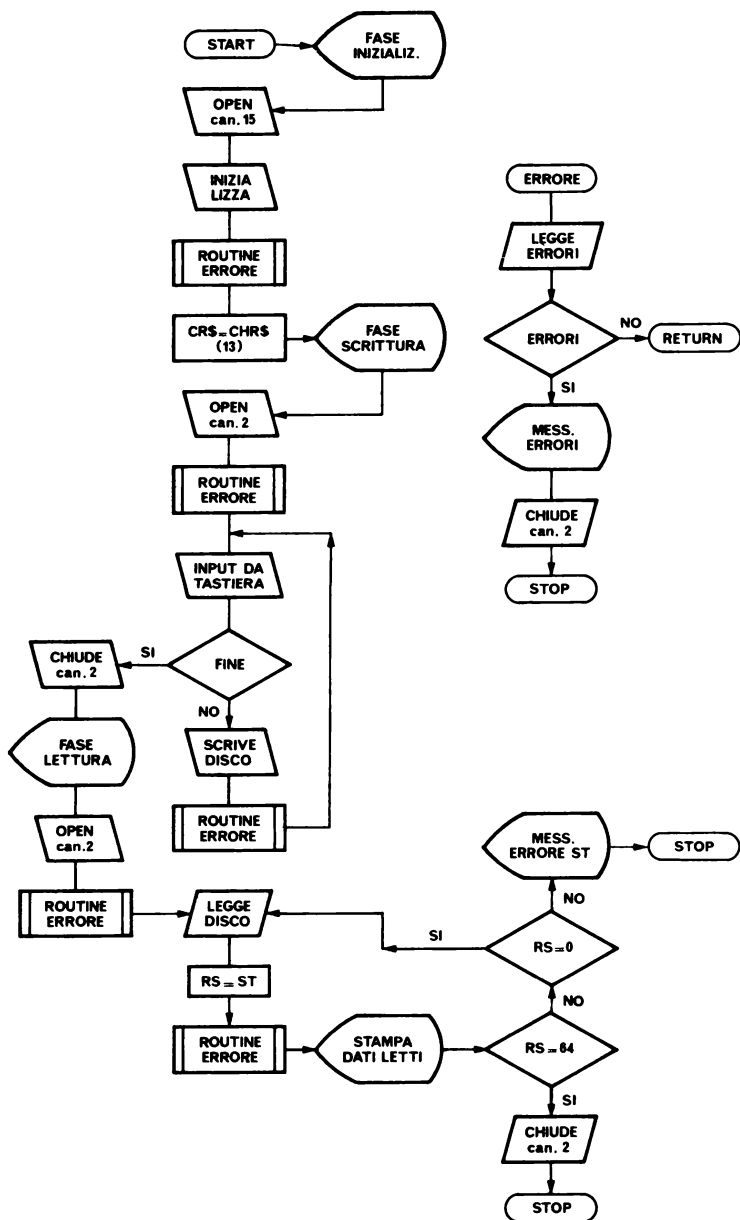



Figura 7-5. Schema a blocchi scrittura/lettura di un FILE sequenziale su disco.

Non potete chiedere D$ senza chiedere anche le prime 3, ma potete chiedere o solo la prima, o le prime due, o le prime tre, o tutte.

ESEMPIO 10 OPEN 1,8,15
20 INPUT # 1,A$,B$,C$,D$
30 PRINT A$,B$,C$,D$

con questo programma leggete dal canale 15 l'errore verificatosi, lo evidenziate al video e spegnete la luce di errore. Si possono usare anche variabili senza il simbolo dollaro, cioè variabili numeriche. Nella Appendice B riportiamo l'elenco degli errori disco che si possono avere.

Ora, come applicazione di quanto studiato, vediamo un semplice programma che scrive un file sequenziale di prova e poi lo rilegge o lo stampa al video. I dati da scrivere sul disco vengono richiesti alla tastiera e per chiudere l'ingresso dei dati si scrive la parola "FINE". Nel programma è compresa l'inizializzazione iniziale del disco e la routine di errore. Inoltre viene analizzata in lettura la parola di stato ST per vedere se il file è finito, condizione di END OF FILE, e per vedere se ci sono stati errori.

LISTA DEL PROGRAMMA SEQ1

```
1 REM     ESEMPIO DI SCRITTURA E LETTURA DI UN FILE
2 REM     SEQUENZIALE DI DATI USANDO IL DISCO 0
5 REM
10 PRINT"FASE INIZIALIZZAZIONE DISCO"
20 OPEN 15,8,15:REM            OPEN IL CANALE 15 DEI COMANDI
30 PRINT 15,"I0":REM           INIZIALIZZA IL DISCO 0
40 GOSUB 1000:REM              LEGGE IL CANALE ERRORI
50 CR$=CHR$(13):REM            PONE CR$=AL CARATTERE CARRIAGE RETURN
60 PRINT"FASE SCRITTURA FILE"
70 OPEN 2,8,2,"@0:FILE DI PROVA,S,W":REM      APRE IL FILE LOGICO 2
80 REM                                        SUL CANALE 2 PER RISCRIVERLO
90 REM                                        IL FILE SI CHIAMA FILE DI PROVA
100 REM                                       ED E' UN FILE SEQUENZIALE
110 GOSUB 1000:REM                 LEGGE IL CANALE ERRORI
120 INPUT"A$,B";A$,B:REM           LEGGE UN NOME ED UN NUMERO
130 IF A$="FINE" THEN 170:REM      PER CHIUDERE SCRIVERE FINE
140 PRINT#2,A$;",";STR$(B);CR$:REM SCRIVE SUL DISCO
150 GOSUB 1000:REM                 LEGGE IL CANALE ERRORI
160 GO TO 120:REM                  TORNA A LEGGERE
170 CLOSE 2:REM                    CHIUDE IL FILE LOGICO 2
180 PRINT"FASE LETTURA FILE"
190 OPEN 2,8,2,"0:FILE DI PROVA,S,R":REM  APRE IL FILE LOGICO 2
200 REM                                   SUL CANALE 2 PER LEGGERE
210 GOSUB 1000:REM          LEGGE IL CANALE ERRORI
220 INPUT#2,A$,B:REM        LEGGE A$ E B(STRINGA E NUMERO)
230 RS=ST:REM               PONE LA PAROLA DI STATO ST IN RS
240 GOSUB 1000:REM          LEGGE IL CANALE ERRORI
250 PRINT A$,B:REM          STAMPA QUANTO HA LETTO
260 IF RS=64 THEN 300:REM   SE END OF FILE VA A 300
270 IF RS<>0 THEN 400:REM   TEST SE ERRORI IN PAROLA STATO
280 GO TO 220:REM           TORNA A LEGGERE
300 CLOSE 2:REM             CHIUDE FILE TERMINATO
305 CLOSE 15                CHIUDE CANALE COMANDI
310 STOP
400 PRINT"ERRORE PAROLA STATO=";RS:REM        EVIDENZIA PAROLA STATO
410 STOP
```

```
1000 REM                    SOTTOPROGRAMMA ANALISI ERRORI
1010 INPUT#15,EN$,EM$,ET$,ES$:REM     LEGGE IL MESSAGGIO ERRORE
1020 REM                       EN$=NUMERO ERRORE
1021 REM                       EM$=MESSAGGIO ERRORE
1022 REM                       ET$=TRACCIA O 00
1023 REM                       ES$=SETTORE O 00
1025 IF EN$="00" THEN RETURN:REM      RITORNA AL PROGRAMMA SE NO ERRORI
1030 PRINT"ERRORE IN OPERAZIONI DISCO"
1040 PRINT EN$,EM$,ET$,ES$:REM        STAMPA GLI ERRORI
1050 CLOSE 2:REM                      CHIUDE IL FILE DATI
1160 STOP
```

Come potete vedere dalla lista del programma nella stringa comando della OPEN accetta delle abbreviazioni: S per SEQUENTIAL, U per user, W per WRITE, R per READ.

CAPITOLO 8

USO DEL DOS SUPPORT

Sul disco TEST-DEMO è contenuto un programma di nome DOS SUPPORT 4.0, se si carica nella memoria del PET tale programma, vengono semplificati i comandi per il disco. La procedura da seguire è la seguente:

- montare sull'unità 0 il disco TEST/DEMO,
- scrivere il comando LOAD "*", 8

Questo comando non solo carica in memoria il programma DOS SUPPORT 4.0, ma prima inizializza il disco 0 e provvede ad aprire il filo logico. Il programma viene collocato nella parte più alta della memoria RAM e non viene disturbato dal caricamento di altri programmi in memoria. Dopo aver caricato in memoria il programma DOS SUPPORT, si deve dare il RUN per renderlo efficiente. Notate che se in seguito darete ancora il comando LOAD "*", 8 verrà caricato in memoria l'ultimo programma per il quale si è avuto o un LOAD o un SAVE. Il fatto di avere in memoria funzionante il DOS SUPPORT, fa si che non si deve più usare il comando PRINT # per dare comandi disco. Si deve solo far precedere il comando da uno di questi due simboli:

>@

Quando si danno in questo modo i comandi disco, non è necessario usare la OPEN, ci pensa il programma DOS ad aprire i files.
Esempi:

> I0 equivale a PRINT#1, "I0"
@ S0:FILE1 equivale a PRINT#15, "S0:FILE1"

Per caricare in memoria la DIRECTORY del disco si usa LOAD "$0", 8 (per il disco 0) e con questo comando si distrugge il contenuto della memoria. Con il nuovo metodo si può scrivere:

> $0 ed la DIRECTORY del disco 0 è evidenziata solo allo schermo senza distruggere il contenuto della memoria.

Scrivendo: >$1:Q* si ottiene sullo schermo l'elenco di tutti i files del disco 1 che iniziano per Q.
Se si vuole fermare lo schermo, premere la barra di spaziatura. Per farlo avanzare ancora schiacciare un qualunque tasto. Per fermare il display della DIRECTORY basta premere RUN/STOP ed il sistema torna al BASIC.

Con uno dei due simboli visti si possono anche richiedere gli errori disco. Basta scrivere il simbolo.

> o@equivale al programmino:

```
OPEN 2, 8, 15
INPUT #2, A$, B$, C$, D$
PRINT A$, B$, C$, D$
```

Per caricare un programma da disco si può scrivere:

/nome-programma

il sistema cerca il programma in ambedue i dischi e lo carica in memoria.

Se si vuole si può usare la (freccia in su) ↑ per caricare un programma e farlo anche partire (non si dà il RUN).

↑ nome-programma

cerca nei due dischi il programma, lo carica e lo fa partire.

La limitazione che si ha nell'uso del DOS SUPPORT è che si può usare solo dando comandi in modo diretto e non in un programma.
Se si dà un reset al PET si deve ricaricare dal disco il DOS SUPPORT.

CAPITOLO 9

SISTEMA AVANZATO DI PROGRAMMAZIONE CON IL DISCO

9.1. Il modo di operare del Sistema Operativo DOS

In questo Capitolo si trattano le informazioni che riguardano la struttura del DOS ed i *comandi di utilità del disco.* Questi ultimi forniscono al programmatore funzioni a basso livello che possono essere usate per speciali applicazioni, come routine speciali di manipolazione del disco e tecniche di accesso casuale. L'interfaccia di controllo del DOS file è responsabile della manipolazione di tutte le informazioni fra il controller del disco e il BUS IEEE-488. Il sistema dei files è organizzato in canali. I canali sono 16. Il canale 15 è per i comandi e gli errori, i canali da 2 a 14 per i files di dati, il canale 0 per il LOAD ed il canale 1 per il SAVE. Ciascun canale è aperto (collegato) attraverso la frase BASIC OPEN. Al momento della OPEN il DOS assegna uno spazio di lavoro al canale ed adibisce una o due aree BUFFER di I/O per il disco. Se manca o lo spazio di lavoro o il buffer, si ha errore NO CHANNEL. Un'area BUFFER di 256 bytes è la memoria comune tra il controller del disco e l'interfaccia di conrollo del file. Dei 16 possibili buffer, 3 sono usati dal DOS per: BLOCK AVAILABILITY MAP (BAM), spazi per le variabili, canali di comando per I/O e coda di lavoro del controller del disco. La coda di lavoro è il più importante collegamento tra i due controller. Le operazioni sono iniziate da parte del file inviando al controller del disco le informazioni riguardo l'indirizzo del settore ed il tipo di operazione. Il controller del disco cerca quale operazione è meglio lanciare e provvede, dopo di che invia la condizione di eventuale errore. Se l'operazione non va a buon fine essa viene ritentata da parte del file un certo numero di volte, tale numero dipende dal tipo di operazione, prima di inviare al PET un messaggio di errore. L'indirizzo secondario della OPEN viene usato come numero del canale.

Sul disco si hanno 35 tracce numerate da 1 a 35. Le tracce sono divise in settori, ma il numero di settori per traccia non è costante; si ha:

- tracce da 1 a 17 con 21 settori numerati da 0 a 20,
- tracce da 18 a 24 con 20 settori numerati da 0 a 19.
- tracce da 25 a 30 con 18 settori numerati da 0 a 17
- tracce da 31 a 35 con 17 settori numerati da 0 a 16

Si hanno quindi in totale 690 settori di 256 bytes ciascuno. La traccia 18 è usata dal sistema.

9.2. Speciali OPEN e CLOSE per accesso diretto

Con la OPEN si può chiedere di adibire un generico buffer al canale o uno specifico buffer al canale:

OPEN 2,8,4,"#" assegna un qualunque buffer disponibile al canale 4 per operazioni sui files

OPEN 2,8,4,"#12" tenta di assegnare il buffer 12 al canale 4

Se, nel primo caso, mancano buffer, o nel secondo il buffer 12 è già occupato si ha l'errore NO CHANNELS. Se si vuole sapere quale buffer è stato allocato con la OPEN, si può usare il comando GET #. Il byte trasmesso è il numero del buffer allocato. Tale operazione va fatta solo prima di eseguire o una operazione di lettura o una operazione di scrittura, relativa a quel file.

La frase CLOSE già nota chiude il canale, libera il buffer e trascrive la BAM sul disco usato. Si raccomanda di usare solo un disco per accesso diretto onde evitare possibili confusioni.

Usare il disco per accesso diretto significa creare files senza la normale registrazione nella directory. Tali files possono essere copiati solo con il comando DUPLICATE o con un programma scritto dall'utente, non con il COPY che richiede il nome del file.

9.3. Comandi di utilità del disco

Sono disponibili i comandi riportati nella tabella seguente:

Comando	Abbreviazione comando	Sintassi comando
BLOCK-READ	B-R	"B-R:SA,DR,T,S"
BLOCK-WRITE	B-W	"B-W:SA,DR,T,S"
BLOCK-EXECUTIVE	B-E	"B-E:SA,DR,T,S"
BUFFER-POINTER	B-P	"B-P:SA,P"
BLOCK-ALLOCATE	B-A	"B-A:DR,T,S"
BLOCK-FREE	B-F	"B-F:DR,T,S"
*MEMORY-WRITE	M-W	"M-W"ADL/ADH/NC/ DATA
*MEMORY-READ	M-R	"M-R"ADL/ADH
*MEMORY-EXECUTE	M-E	"M-E"ADL/ADH
USER	U	"Ui:parametri"

Nota: ADL, ADH e NC sono singoli bytes e quindi per ottenere il valore si deve usare CHR$ (n), n = indirizzo del byte. Si presuppone che ogni comando sia preceduto da PRINT#.La ricerca dei parametri dei comandi viene fatta partendo dai due punti (:) o dal quarto carattere, se mancano i due punti.

Esempi di scrittura validi per lo stesso comando:

"BLOCK-READ:2,1,4,0" — questo comando legge dal canale 2 sul disco 1 dalla traccia 4 il settore 0.

"B-R 2,1,4,0"
"B-R";2;1;4;0
"B-READ";2;1;4;0

nella tabella:

SA è l'indirizzo secondario della OPEN
DR è 0 per il disco 0 e 1 per il disco 1
T è il numero della traccia da 1 a 35
S è il numero del settore da 0 a 20 per tracce 1/17
0 a 19 per tracce 18/24
0 a 17 per tracce 25/30
0 a 16 per tracce 31/35
P è la posizione del puntatore nel buffer
ADL è l'indirizzo del byte più basso interessato all'operazione
ADH è l'indirizzo del byte pù alto interessato all'operazione
NC è il numero di caratteri coinvolti, da 1 a 34
DATA è il numero di bytes di dati, al massimo 34
i è l'indice per la TABLE USER
parametri (opzionali) sono i parametri associati con il comando U.

Per i tre comandi segnati con asterisco è accettata solo la forma abbreviata, per gli altri sono accettate le due forme.
All'interno degli apici, dopo le parole chiave, ci vogliono i due punti, gli altri dati possono essere separati sia da virgola che da spazio che da SKIP (→). Fuori dagli apici i caratteri devono essere separati, se si vuole separarli, dal punto e virgola.

9.4. Descrizione dei comandi di utilità del disco

BLOCK-READ. Permette un accesso diretto a qualunque blocco sui due dischi. Usato insieme ad altri comandi di questo tipo, permette di creare per mezzo del BASIC l'accesso casuale ai files. L'indicatore di ultimo carattere viene trovato nella posizione zero del blocco; se questo carattere è individuato tramite un comando GET # o INPUT #, viene inviata la segnalazione di fine identificazione, allora l'operazione termina e viene mezzo 64 in

ST. Esempio: leggere il blocco dal disco 1, traccia 18, settore 0; nell'area buffer del canale 5, si può scrivere:

"BLOCK-READ:5,1,18,0" oppure
"B-R5,1,18,0" oppure
"B-R"5;1;18;0

BLOCK-WRITE. Quando viene eseguito, il pointer del buffer in uso è utilizzato come pointer dell'ultimo carattere ed è messo nella posizione zero del blocco. Il blocco è scritto sul disco ed il puntatore del buffer è messo in posizione 1. Esempio: "B-W";7;0;35;10 - scrive il buffer del canale 7 sul blocco del disco 0, traccia 35, settore 10.

BLOCK-EXECUTE. Questo comando permette che parte del sistema operativo risieda sul disco. Esso legge un blocco in memoria e va ad eseguire il contenuto del blocco. Il blocco deve contenere istruzioni in linguaggio macchina.

BUFFER-POINTER. Questo comando modifica il contenuto del puntatore associato con un dato canale. Esso serve per accedere a particolari record o campi all'interno di un blocco. Esempio: "B-P«";2;1 - mette il puntatore del buffer associato al canale 2 all'inizio dei dati (posizione 1).

BLOCK-ALLOCATE. Si può creare una BAM appropriata nella memoria DOS per indicare che un blocco è usato. Il sistema non potrà usare questo blocco in altre operazioni sequenziali. La BAM creata sarà scritta su disco quando si chiude il canale di scrittura o il canale di accesso diretto. Se il blocco indicato è già stato usato, si avrà un errore di canale che indicherà il successivo blocco disponibile se c'è oppure 0 con il messaggio NO BLOCK.

BLOCK-FREE. La BAM viene aggiornata in memoria per indicare come libero il blocco indicato. Il disco viene aggiornato quando si chiude.

MEMORY-WRITE. Questo comando permete l'accesso diretto alla memoria a livello di BYTE. Si può usare quando si programma in linguaggio machina. Per usare il comando in BASIC bisogna servirsi della funzione CHR$ per passare i parametri. Con una operazione di questo tipo si possono trasferire in memoria fino a 34 byte. L'indirizzo del byte più basso deve precedere quello del byte più alto.

MEMORY-READ. Il comando rende disponibile il byte puntato dalla stringa di comando. Il successivo GET # dal canale 15 trasmette questo byte. Si possono leggere le variabili DOS o i contenuti dei buffer. Non si può usare un INPUT # dopo questo comando fino a dopo l'uso di altri comandi DOS non di questo tipo.

MEMORY-EXECUTE. Con questo comando si possono far eseguire pezzi di programmi memorizzati in locazioni conosciute. Le routine devono termi-

nare con RTS $60 per restituire il controllo al DOS alla fine. Questi comandi non aggiornano il canale di errore.

USER. Questo comando permette il collegamento a qualsiasi codice macchina attraverso la tavola dei salti (TABLE USER) indirizzata da uno speciale puntatore. L'indice alla tavola è il parametro "i" indicato nella stringa dei comandi. Possono essere usati i caratteri ASCII da 0 a 9 oppure da A a I (per 1/9) e J per 0. Lo 0 o J posiziona ad una tavola di salti speciali per collegamenti a routine speciali. Il comando U1 equivale a BLOCK-READ e si può usare in alternativa. Il comando U2 equivale a BLOCK-WRITE solo che scrive sul disco senza modificare il contenuto della posizione 0 come fa B-W. Questo può essere utile se si vuole leggere un blocco con B-R, aggiornarlo e riscriverlo con B-P e PRINT #.

La tavola dei salti

Prima designazione	Seconda designazione	Funzione
U1	UA	equivale a BLOCK-READ
U2	UB	equivale a BLOCK-WRITE con una differenza
U3	UC	Salto a $1300
U4	UD	Salto a $1303
U5	UE	Salto a $1306
U6	UF	Salto a $D008
U7	UG	Salto a $D00B
U8	UH	Salto a $D00E
U9	UI	Salto a $D0D5
U0	UJ	Power up $E18E

9.5. Esempi di uso dei files

Sul disco TEST/DEMO sono contenuti due programmi di nome:

- SEQUENTIAL 1.00
- RANDOM 1.00

è consigliabile, servendosi della lista di questi due programmi, studiare attentamente come si gestiscono i files. Il primo riguarda una gestione sequenziale, mentre il secondo riguarda una gestione random. La lista del programma SEQUENTIAL 1.00 è riportata nel paragrafo 7.14 con i commenti. La lista del programma RANDOM viene riportato alla fine di questo paragrafo. Per quest'ultimo programma riportiamo un breve

commento. Dato che il comando BLOCK-ALLOCATE restituisce attraverso il canale di errore il successivo blocco disponibile sul disco, esso può essere utilizzato per allocare le registrazioni. Con questa possibilità si possono creare file random, pur senza conoscere l'effettiva struttura del disco. Bisogna però, per poter utilizzare un programma BASIC, tener nota dei blocchi usati.

L'esempio RANDOM 1.00 dà modo di capire quali sono i comandi necessari per accedere ai blocchi. Si noti che vengono usati i comandi U1 e U2. È necessario usare questi comandi poichè vengono memorizzati più record in un blocco ed è necessario manipolare in BASIC i puntatori di fine record. In applicazioni di piccola mole può essere vantaggioso usare i comandi B-R e B-W. Si usa uno schema a record e si può accedere al singolo record. Le linee sotto 2000 riguardano l'accesso a ciascun record. Le routine di accesso ai campi appoggiano a sinistra dati binari o alfabetici, e a destra dati numerici normali. In una situazione reale il programma dovrebbe generare dei messaggi di errore per l'operatore, come pure intraprendere azioni correttive. Dovrebbe anche essere possibile eseguire il SORT dei dati (cioè metterli in un certo ordine voluto) o aggiungere campi chiave al programma. La dimensione dei record, compresi gli indicatori di campo, dovrebbe essere minore di 254 caratteri. Le dimensioni dei campi sono al massimo di 80 caratteri per le limitazioni imposte dal comando INPUT del BASIC.

Per realizzare l'accesso casuale al file vengono usati due file sequenziali come indici. Ciascuno di essi contiene il nome assegnato con il codice creato (linee 1100-1180) più una estensione di 6 caratteri. Poichè i nomi dei files primari possono essere formati da sequenze fino a 10 caratteri, essi sono eventualmente completati con spazi. I due files sequenziali si chiamano: FILENAME.DESCR e FILENAME.KEY01. Il file con estensione .DESCR contiene le informazioni relative alla locazione ed alla struttura dei record; quello con estensione .KEY01 contiene il primo campo di ciascun record ed il relativo numero di record. Nell'esempio i record ai quali si vuole accedere in modo casuale possono risiedere su un disco diverso da quello su cui si trovano i due files sequenziali. Il codice OPEN (linee 1200-1275) richiede l'ID del disco su cui si trova il file RANDOM a scopo di controllo.

Programma RANDOM

```
1 REM RANDOM 1.0
2 REM SUBROUTINES PER LA GESTIONE DI FILES AD ACCESSO CASUALE
3 REM  LE VARIABILI SONO SCELTE DAI PARAMETRI DELLA DESCRIZIONE DEL FILE
4 REM E DALLA LISTA DELLE CHIAVI DEI FILES DEFINITI DAL PROGRAMMA DELL'UTENTE
5 REM LE VARIABILI DEVONO DESCRIVERE LA STRUTTURA DEL FILE
6 REM TUTTE LE FUNZIONI PRENDONO COME PUNTO DI PARTENZA
7 REM LE VARIABILI DEFINITE DI SEGUITO
10 REM
11 REM **************************
12 REM
13 POKE1022,128:REM ESCLUDE IL SUPPORTO DOS
15 M$=CHR$(13):REM INDICATORE DI FINE CAMPO
16 SP$="                                        "+"":REM SPAZI DI RIEMPIMENTO
20 C0=2:          REM CANALE DIRETTO
21 C1=3:          REM CANALE SEQUENZIALE
25 CC=15:         REM CANALE DI COMANDO
30 D=0:           REM # DRIVE IN USO
31 T=0:           REM # TRACCIA IN USO
32 S=0:           REM # SETTORE IN USO
35 DD=0:          REM # DESCRITTORE DRIVE
36 RD=0:          REM # DRIVE RANDOM
40 ID$="":        REM IDENTIFICATORE DEL DISCO RANDOM
45 NR=0:          REM # RECORDS NEL FILE RANDOM
46 CR=0:          REM # RECORD IN USO
47 FR=0:          REM PRIMO RECORD DISPONIBILE
50 NF=0:          REM # CAMPI NEL RECORD
51 CF=0:          REM # CAMPO IN USO
55 RB=0:          REM # RECORDS PER BLOCCO
56 RS=0:          REM DIMENSIONE DEL RECORD IN BYTES
60 NB=0:          REM # BLOCCHI NEL FILE RANDOM
65 E=0:           REM INDICATORE D'ERRORE (OK=0)
66 REM EN$,EM$,ET$,ES$,ET,ES  VARIABILI DEL CANALE DI ERRORE
70 EP=.5/256:  REM CORREZIONE PER NUMERI INTERI
75 AS=0:          REM MODO D'INDIRIZZAMENTO TABELLE
76 REM AS=0: USO INDICE TABELLE; AS=1: T&S SONO INIZIALIZZATI
77 REM CR = DISPOSIZIONE RECORD NEL BLOCCO
90 REM  LE VARIABILI IN "A" SONO TEMPORANEE
95 DN=8:OPENCC,DN,CC:  REM DN= NUMERO DELLA DEVICE
98 GOTO2000:   REM INIZIO DEL PROGRAMMA DELL'UTENTE
99 REM
100 REM **************************
101 REM ROUTINE DI DIMENSIONAMENTO DEL FILE RANDOM
102 REM  PRIMA ASSEGNAZIONE NR, NF & NB
103 REM
105 GOSUB150
110 IFFP%=-1THENRETURN
111 FP%=-1
115 DIM FS%(NF) :REM DIMENSIONI DEL CAMPO
120 DIM FP%(NF) :REM POSIZIONE DEL CAMPO
125 REM          FP%(I)= SUM [FS%(I-1)]
130 DIM FT%(NF) :REM TIPO DI CAMPO: 0:BINARIO, 1:NUMERICO, 2:ALFANUMERICO
135 DIM FH$(NF) :REM INTESTAZIONE DEL CAMPO
140 DIM F$(NF)  :REM ARGOMENTI CAMPO ALFANUMERICO O BINARIO
145 DIM F(NF)   :REM ARGOMENTI CAMPO NUMERICO
146 RETURN
150 IFIT%=-1THENRETURN
151 IT%=-1
155 DIM IT%(NB) :REM VETTORE DEGLI INDICI DI TRACCIA
160 DIM IS%(NB) :REM VETTORE DEGLI INDICI DI SETTORE
165 DIM K1$(NR) :REM VALORE PRIMARIO DELLA CHIAVE
170 DIM RR%(NR) :REM RELATIVA LISTA DI RECORD PER CARATTERE
175 RETURN
200 REM **************************
201 REM AGGIORNAMENTO RECORD , CR
202 REM
205 GOSUB900
```

```
210 PRINT#CC,"U1:"C0;D;T;S
215 PRINT#CC,"B-P:"C0;RP
220 FORCF=1TONF
225 GOSUB500
230 NEXTCF
235 PRINT#CC,"U2:"C0;D;T;S
240 GOSUB1000:IFETHEN1900
245 RETURN
300 REM *************************
301 REM LETTURA RECORD, CR
302 REM
305 GOSUB900
310 PRINT#CC,"U1:"C0;D;T;S
315 PRINT#CC,"B-P:"C0;RP
320 GOSUB1000:IFETHEN1900
325 FORCF=1TONF
330 GOSUB600
335 NEXTCF
340 RETURN
400 REM *************************
401 REM AGGIORNAMENTO CAMPO(CF) DEL RECORD CR, AGGIORNAMENTO DEL SINGOLO CAMPO
402 REM
405 GOSUB900
410 PRINT#CC,"U1:"C0;D;T;S
415 GOSUB1000:IFETHEN1900
420 PRINT#CC,"B-P:"C0;FP%(CF)+RP
425 GOSUB500 :REM UPDATE FIELD
430 PRINT#CC,"U2:"C0;D;T;S
435 GOSUB1000:IFETHEN1900
440 RETURN
450 REM *************************
451 REM LETTURA DEL CAMPO(CF) DEL RECORD CR, LETTURA DEL SINGOLO CAMPO
452 REM
455 GOSUB900
460 PRINT#CC,"U1:"C0;D;T;S
465 GOSUB1000:IFETHEN1900
470 PRINT#CC,"B-P:"C0;FP%(CF)+RP
475 GOSUB600 :REM READ FIELD
480 RETURN
500 REM *************************
501 REM AGGIORNAMENTO CAMPO(CF), IL B-P E' ATTIVATO
502 REM
510 IFFT%(CF)<>1THEN520
515 A$=RIGHT$(SP$+STR$(F(CF)),FS%(CF)):GOTO530
520 A$=LEFT$(F$(CF)+SP$,FS%(CF))
530 PRINT#C0,A$;M$;
535 RETURN
600 REM *************************
601 REM LETTURA CAMPO(CF),IL B-P E' ATTIVATO
602 REM
610 IF FT%(CF) THEN645
615 A1$=""
620 FORJ=1TOFS%(CF)
625 GET#C0,A$:IFA$=""THENA$=CHR$(0)
630 A1$=A1$+A$
635 NEXT:F$(CF)=A1$
640 GET#C0,A$:RETURN
645 INPUT#C0,F$(CF)
650 IFFT%(CF)<>1THEN RETURN
655 F(CF)=VAL(F$(CF)):RETURN
700 REM *************************
701 REM ALLOCAZIONE DI UN BLOCCO, T & S =TRACCIA E SETTORE RICHIESTI
702 REM  I VALORI DI T & S RITORNATI SONO QUELLI USATI (T=18 VIENE SALTATA)
703 REM
710 GOSUB800:IFETHEN1900: REM CHECK T & S
715 PRINT#CC,"B-A:"D;T;S
720 INPUT#CC,EN,EM$,ET,ES
725 IFEN=0THENRETURN
730 IFEN<>65THEN1900
735 IFET=18THENT=19:S=0:GOTO0715
```

```
736 T=ET:S=ES
740 GOTO715
750 REM *************************
751 REM T & S = TRACCIA E SETTORE DEL BLOCCO LIBERO
752 REM
760 GOSUB800:IFETHEN1900: REM CHECK T & S
770 PRINT#CC,"B-F:"D;T;S
780 INPUT#CC,EN,EM$,ET,ES
785 IFEN=0THENRETURN
790 GOTO1900
800 REM *************************
801 REM CONTROLLO SETTORE MASSIMO
802 REM
810 IFT>35THEN1900
820 E=0:IFT=0THEN EN=40:GOTO1900
840 A3=16:IFT>30THEN880
850 A3=17:IFT>24THEN880
860 A3=19:IFT>17THEN880
870 A3=20
880 IFS>A3THEN1900
890 RETURN
900 REM *************************
901 REM RICERCA LA TRACCIA, IL SETTORE, IL PUNTATORE DEL RECORD
902 REM DALLA TABELLA  DEGLI INDICI
903 REM
905 D=RD
910 E=0
915 IFAS=-1THENRP=CR*RS+1:GOTO950
920 RP=INT((CR-1)/RB+EP):IFRP>NB OR RP<0THENEN=41:GOTO1900
930 T=IT%(RP):S=IS%(RP)
940 RP=INT(((CR-1)/RB-RP+EP)*RS*RB)+1
950 IFRP>254THEN EN=41:GOTO1900
960 RETURN
1000 REM *************************
1001 REM ANALIZZA LO STATO DI ERRORE
1002 REM
1005 INPUT#CC,EN$,EM$,ET,ES
1010 EN=VAL(EN$):E=0
1015 IF EN$="00" THEN RETURN
1017 ET$=STR$(ET):ES$=STR$(ES)
1020 IFEN$<>RIGHT$("0"+MID$(STR$(EN),2),2)THEN1070
1030 IF EN=1 THEN EM$= ET$+" "+EM$: RETURN
1035 E=E+1
1040 EM$="▒"+EN$+"■ "+EM$
1050 IF EN<30 OR EN=65 THEN EM$=EM$+" ON "+ET$+", "+ES$
1060 RETURN
1070 EM$="▒IL SISTEMA NON RISPONDE CORRETTAMENTE"
1080 EM$=EM$+EN$+EM$+ET$+ES$
1085 E=E+1
1090 RETURN
1100 REM *************************
1101 REM CREAZIONE DELLA DESCRIZIONE DEL FILE
1102 REM INPUT: F$= FILENAME
1103 REM         ID$,NR,NF,FS%(),FT%(),FH$()
1104 REM         DD= # DESCRITTORE DEL DRIVE
1105 REM         RD= # DEL DISCO RANDOM
1106 REM I DRIVES DEVONO ESSERE STATI INIZIALIZZATI
1109 REM
1110 RS=1:D=RD
1115 FORA0=1TONF:FP%(A0)=RS:RS=FS%(A0)+RS+1:NEXT:RS=RS-1
1116 RB=INT(254/RS+EP)
1120 OPENC0,DN,C0,"#":GOSUB1000:IFETHEN1900
1121 GOSUB1280
1122 PRINT#CC,"B-P:"C0;1
1123 FORA0=1TORB:FORA1=1TONF
1124 PRINT#C0,LEFT$(SP$,FS%(A1));M$;
1126 NEXTA1,A0
1130 NB=INT(NR/RB+EP):IF(NR/RB-NB)*RB>=1THENNB=NB+1
1135 T=1:S=0:GOSUB150
1140 FORA0=0TONB-1:GOSUB710:IFETHEN1900
```

```
1145 IT%(A0)=T:IS%(A0)=S:GOSUB430:NEXT
1150 GOSUB710
1152 PRINT#CC,"B-P:"C0;1
1155 PRINT#C0,NR;M$;1;M$;NB;M$;RS;M$;RB;M$;NF;M$;
1160 PRINT#CC,"B-W:"C0;D;T;S
1165 A$=STR$(DD)+":"+LEFT$(F$+SP$,10)+".DESCR,U,W"
1166 OPENC1,DN,C1,A$
1167 GOSUB1000:IFETHEN1900
1168 PRINT#C1,ID$;M$;T;M$;S;M$;
1170 FORA0=1TONF:PRINT#C1,CHR$(FS%(A0));CHR$(FT%(A0));FH$(A0);M$;:NEXT
1175 FORA0=0TONB-1:PRINT#C1,CHR$(IT%(A0));CHR$(IS%(A0));:NEXT
1180 CLOSEC1:CLOSEC0:RETURN
1200 REM *************************
1201 REM APERTURA DEL FILE RELATIVO
1202 REM INPUT: F$= NOME DEL FILE
1203 REM         DD= # DESCRITTORE DEL DRIVE DEL FILE
1204 REM         RD= # DEL DRIVE DEL DISCO RANDOM
1205 REM I DRIVES DEVONO ESSERE STATI INIZIALIZZATI
1209 REM
1210 A$=STR$(DD)+":"+LEFT$(F$+SP$,10)+".DESCR,U,R"
1215 OPENC1,DN,C1,A$:GOSUB1000:IFETHEN1900
1220 INPUT#C1,ID$,T,S
1225 OPENC0,DN,C0,"#":GOSUB1000:IFETHEN1900
1226 GOSUB1280
1227 PRINT#CC,"B-R:";C0;RD;T;S:GOSUB1000:IFETHEN1900
1230 INPUT#C0,NR,FR,NB,RS,RB,NF
1235 GOSUB100:FT%(0)=T:FS%(0)=S
1240 FORA0=1TONF:GOSUB1298:FS%(A0)=ASC(A$)
1245 GOSUB1298:FT%(A0)=ASC(A$)
1250 INPUT#C1,FH$(A0):NEXT
1255 FORA0=0TONB-1:GOSUB1298:IT%(A0)=ASC(A$)
1260 GOSUB1298:IS%(A0)=ASC(A$):NEXT
1265 GOSUB1000:IFETHEN1900
1270 CLOSEC1
1275 RETURN
1280 PRINT#CC,"U1:";C0;RD;",18,0":GOSUB1000:IFETHEN1900
1285 PRINT#CC,"B-P:";C0;162
1286 GET#C0,A$,A1$:A$=A$+A1$:IFID$<>A$THENEN=43:EM$="DISCO SBAGLIATO":GOTO1900
1290 RETURN
1298 GET#C1,A$:IFA$=""THENA$=CHR$(0)
1299 RETURN
1400 REM *************************
1401 REM CHIUSURA DEL FILE RELATIVO
1402 REM INPUT: LE VARIABILI DELLA OPEN DEVONO ESSERE VALIDE
1409 REM
1410 PRINT#CC,"B-P:"C0;1
1420 PRINT#C0,NR;M$;FR;M$;NB;M$;RS;M$;RB;M$;NF;M$;
1430 PRINT#CC,"B-W:"C0;D;FT%(0);FS%(0)
1440 CLOSEC0
1490 RETURN
1900 E=E+1:RETURN
2000 INPUT"███DESIDERI CREARE UN FILE  N███";A$:IFLEFT$(A$,1)<>"S"THEN2100
2001 INPUT"█NOME DEL FILE RANDOM";F$
2002 INPUT"NUMERO DELLA CHIAVE DEL FILE";DD
2003 INPUT"NUMERO DEL DRIVE PER IL FILE RANDOM";RD
2005 INPUT"SCRIVI L'ID DEL DISCO RANDOM   __█████";ID$:ID$=LEFT$(ID$,2)
2006 INPUT"NUMERO DEI RECORDS";NR
2007 INPUT"NUMERO DEI CAMPI PER RECORD";NF
2010 GOSUB110
2015 PRINT"█  SCRIVI IL NOME DEL CAMPO,  LA MISURA, IL TIPO: "
2016 PRINT"0=BINARIO, 1=NUMERICO, 2=ALFANUMERICO█"
2019 RS=0
2020 FORI=1TONF:PRINT"CAMPO";I,:INPUTFH$(I),FS%(I),FT%(I):RS=FS%(I)+RS+1:NEXT
2025 A$="I":IFDD=RDTHENA$="I"+STR$(DD)
2030 PRINT#CC,A$
2040 GOSUB1100:IFETHEN3900
2050 OPEN4,8,4,STR$(DD)+":"+LEFT$(F$+SP$,10)+".KEY01,U,W"
2055 PRINT#4,0;M$;:CLOSE4
2090 GOTO2120
2100 REM APRE IL FILE RANDOM PER L'ACCESSO
```

```
2103 INPUT"▒NOME DEL FILE RANDOM";F$
2105 INPUT"NUMERO DELLA CHIAVE DEL FILE";DD
2110 INPUT"NUMERO DEL DRIVE PER IL FILE RANDOM";RD
2120 GOSUB1200:IFETHEN3900
2140 OPEN4,8,4,STR$(DD)+":"+LEFT$(F$+SP$,10)+".KEY01,U"
2142 INPUT#4,RR:IFRR=0THEN2147
2145 FORI=1TORR:INPUT#4,K1$(I),RR%(I):NEXT
2147 CLOSE4
2150 PRINT"▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒ESEMPIO DI ACCESSO CASUALE▒"
2155 PRINT"▒PREMERE // PER USCIRE▒"
2156 PRINT"▒SCHIACCIARE RETURN PER AGGIUNGERE RECORD"
2157 PRINT"▒BATTERE /DIR PER VEDERE LA LISTA"
2160 PRINT"▒▒QUALE RECORD VUOI"
2161 INPUT"VEDERE    ▒▒▒";RR$
2165 IFRR$=" "THEN2310
2167 IFRR$="//"THEN2400
2168 IFRR$="/DIR"THENGOSUB4000:GOTO2160
2170 FORII=1TORR:IFK1$(II)<>RR$THENNEXT:GOTO2300
2175 CR=RR%(II):GOSUB300
2180 FORI=1TONF:PRINTI;")"FH$(I)":",F$(I):NEXT:PRINT
2185 FF=0
2190 INPUT"QUALCHE MODIFICA    ▒▒▒";A$:IFLEFT$(A$,1)<>"S"THEN2220
2195 INPUT"IN QUALE CAMPO";A
2200 PRINT"   "F$(A):PRINT"▒";:INPUTF$(A):F(A)=VAL(F$(A))
2210 FF=1:GOTO2190
2220 IFFF=0THEN2160
2222 IFA=1THENK1$(II)=F$(A)
2225 GOSUB200
2230 GOTO2160
2300 PRINT"▒RECORD NON PRESENTE"
2305 INPUT"VUOI AGGIUNGERE RECORD";A$:IFLEFT$(A$,1)<>"S"THEN2160
2310 PRINT"▒**** AGGIUNTA RECORD ****▒▒"
2312 IFFR>NRTHEN2500
2315 CR=FR:FR=FR+1:RR=RR+1
2320 FORI=1TONF:PRINTFH$(I);:INPUTF$(I):F(I)=VAL(F$(I)):NEXT
2330 GOSUB200
2340 K1$(RR)=F$(1):RR%(RR)=CR
2350 GOTO2160
2400 REM CHIUSURA DEL FILE RANDOM
2405 GOSUB1400
2410 OPEN4,8,4,"@"+STR$(DD)+":"+LEFT$(F$+SP$,10)+".KEY01,U,W"
2420 GOSUB1000:IFETHEN3900
2430 PRINT#4,RR;M$;
2440 FORI=1TORR:PRINT#4,K1$(I);M$;RR%(I);M$;:NEXT
2445 GOSUB1000:IFETHEN3900
2450 CLOSE4
2455 GOSUB1000:IFETHEN3900
2490 POKE1022,8:END:REM ABILITA IL SUPPORTO DOS
2500 PRINT"IL FILE E' PIENO, NON POSSONO ESSERE    AGGIUNTI ALTRI RECORDS"
2510 GOTO2160
3900 PRINTE,EM$:STOP
4000 FORDI=0TONR:PRINTK1$(DI):NEXT:RETURN
```

CAPITOLO 10

IL LINGUAGGIO MACCHINA

10.1. Introduzione

Questo capitolo è dedicato ai lettori che, dopo aver imparato a programmare benissimo il calcolatore in BASIC, desiderano approfondire la conoscienza della programmazione. Per programmare in linguaggio macchina bisogna sapere come funziona il calcolatore e bisogna conoscere i metodi di indirizzamento ed il gruppo di istruzioni in linguaggio macchina di cui il calcolatore dispone. Programmare direttamente in linguaggio macchina significa scrivere in codice esadecimale la sequenza di istruzioni, facendo riferimento ad indirizzi di memoria effettivi per le costanti e le aree di lavoro. Si tratta quindi di un lavoro molto specializzato e per il quale si richiede una buona dose di pazienza. È comunque consigliabile affrontare questa fatica, se si vuole approfondire la conoscenza della programmazione. Inoltre, dopo questa esperienza, si sarà certamente molto grati ai gruppi di lavoro che hanno messo a punto il linguaggio BASIC!

I vantaggi del linguaggio macchina sono essenzialmente quelli di avere programmi molto veloci e che occupano poca memoria. Inoltre, in alcuni casi, come collegamento di terminali speciali al calcolatore, potrà essere necessario ricorrere a brevi routine in linguaggio macchina per risolvere problemi di input o output. Qualora si desideri scrivere programmi lunghi e complessi in linguaggio macchina, si ricorre a linguaggi simbolici a basso livello, tipo *assembler.* In tale caso si programma in codice simbolico mnemonico e si usa un programma traduttore, che si chiama *assemblatore,* per ottenere il codice esadecimale da caricare in memoria. Linguaggi simbolici a basso livello, significa che il rapporto tra istruzione simbolica ed istruzione macchina è di uno a uno. In BASIC una frase simbolica corrisponde ad un buon numero di istruzioni in codice macchina, e quindi può essere classificato come un linguaggio interpretativo ad alto livello. Non si intende qui esporre la programmazione in linguaggio macchina del microprocessore 6502; l'argomento è molto vasto e gli interessati possono servirsi per approfondire l'argomento de "La programmazione del 6502" di Rodnay Zaks, edito pure dal Gruppo Editoriale Jackson.

Scopo di questo capitolo è di suggerire come introdurre in memoria un programma in linguaggio macchina e come eseguirlo. Si possono usare tre metodi. Il primo è servirsi delle possibilità che offre il BASIC sfruttando i comandi POKE,PEEK,USR e SYS. Il secondo è servirsi di un Monitor, che è stato messo a punto dalla Commodore, ed il cui compito è di scrivere bytes in memoria, di esaminarli e di saltare ad eseguire pezzi di programma in codice macchina. Il terzo metodo, di cui diamo qui solo brevi cenni, è di intervenire sul sistema operativo del calcolatore aggiungendo nuove parti in punti opportuni. Si può, per esempio, aggiungere una nuova parte alla subroutine di servizio degli interrupt, che è chiamata sessanta volte al secondo dal segnale di scansione delle interruzioni prioritarie. Oppure si può aggiungere parte di codice alla routine CHARGOT che preleva le linee BASIC dalla memoria prima della loro esecuzione da parte dell'interprete. Si raccomanda comunque di procedere con molta cautela, dal momento che, quando si esce dal BASIC, vengono meno le protezioni che il sistema offre.

10.2. Metodo BASIC

La prima cosa da fare è scrivere un programma in codice simbolico del linguaggio macchina del microprocessore 6502, facendolo terminare con l'istruzione RTS, che serve per rientrar da un sottoprogramma. Prima di scrivere questo programma si deve decidere se verrà richiamato con USR o con SYS, che, come si vedrà più avanti, offrono possibilità diverse. Quando il programma è pronto si deve tradurre in una sequenza di bytes con valore decimale, passando attraverso la codifica esadecimale, se risulta più comodo. La sequenza decimale del programma può essere caricata in memoria servendosi della POKE I,N; dove I è l'indirizzo di un byte ed N è il valore decimale da caricare. Il problema è decidere quale valore usare per I, cioè dove mettere in memoria il programma. Se il sistema non usa la seconda cassetta, si può usare il buffer della seconda cassetta, che si trova agli indirizzi 826/1017. Se il sistema non usa alcuna cassetta, si può usare l'area dei due buffers da 634 a 1017.

Un altro modo può essere quello di modificare a programma, subito all'inizio, il contenuto del puntatore alla fine della memoria; abbassando tale valore si riserva un blocco di memoria nella parte più alta. Altro modo può essere quello di aumentare, subito all'inizio, il valore del puntatore all'inizio della zona variabili, riservandosi così un blocco di memoria tra la fine del programma BASIC (la cui lunghezza deve essere peraltro calcolabile) e l'inizio della zona variabili. Trovato il posto in memoria si deve decidere come caricare il programma con le istruzioni POKE. La cosa più semplice è fare una sequenza di POKE, tante quante sono i bytes da caricare. Altrimenti si possono memorizzare con dei DATA i valori deci-

mali dei bytes e poi con un ciclo FOR/NEXT caricare tutti i bytes usando READ e POKE.

Il comando SYS si scrive: SYS (indirizzo), fa saltare ad eseguire codice macchina memorizzato a "indirizzo", naturalmente l'indirizzo deve essere minore di 65535. Quando si incontra l'istruzione RST il controllo ritorna alla linea BASIC seguente quella del comando SYS. Per ottenere che il codice macchina lavori su variabili trattate e trattabili dal BASIC si deve procedere così:

- scrivere con la POKE le variabili in posizioni predeterminate di memoria byte dopo byte (ricordando la formuletta per dividere in due bytes un numero che ne occupa globalmente due), prima di usare la SYS;
- riottenere con la PEEK i risultati dei calcoli condotti a termine dal codice macchina, andandoli a leggere da predeterminate posizioni, dove li ha memorizzati il programma in codice macchina.

In sostanza il comando SYS non consente di trasmettere parametri al codice macchina, se non servendosi di posizioni di memoria predeterminate.

La funzione USR si scrive: USR (argomento), e fa saltare ad un programma in codice macchina, il cui indirizzo sia stato preventivamente caricato nelle posizioni di memoria 1 e 2 della RAM. Questa funzione consente di passare al sottoprogramma un parametro, "argomento". Tale parametro viene sistemato nell'accumulatore FAC (occupante sei bytes nelle locazioni 94/99 decimali e quindi 5E/63 esadecimali) dal BASIC. Il programma in codice macchina deve lavorare usando il contenuto di FAC e ritornare il risultato dei calcoli in FAC; il contenuto di FAC viene ritornato dal BASIC all'utente. Per esempio:

10 A = USR(75) con questa frase viene mandato in onda il programma il cui indirizzo si trova in 1 e 2; tale programma lavora sul numero 75 e ritorna al programma BASIC in A il risultato dei calcoli. Se il codice macchina non modifica il contenuto di FAC, nell'esempio si avrà A = 075. Comunque prima di usare USR, si devono caricare i due bytes 1 e 2 con l'indirizzo del programma in codice macchina, usando la POKE. Se, per esempio si vuole far patire un codice memorizzato a partire dall'indirizzo 826, si ha:

$(826)_{10} = (033A)_{16}(3A)_{16} = (58)_{10}(03)_{16} = (3)_{10}$ e quindi
POKE 1,58
POKE 2,3 infatti $3 \cdot 256 + 58 = 826$

Il FAC mantiene i numeri floating point in questo modo:

byte 94 esponente con segno, con il sistema di sommare 128 all'esponente che può essere positivo o negativo, ma in valore assoluto minore di 127. Tale esponente ha questi valori per la base binaria; in decimale si ha un campo di variabilità

corrispondente tra 10^{-38} e 10^{+38};

byte 95 cifre più significative della mantissa, sempre normalizzata con il primo bit a 1;

byte 96 ancora mantissa;

byte 97 ancora mantissa

byte 98 cifre meno significative mantissa;

byte 99 segno della mantissa: 0 per positivo e —1 per negativo.

10.3. Metodo Monitor

Il monitor messo a punto dalla Commodore si chiama TIM (Terminal Interface Monitor). Nelle nuove macchine è disponibile in ROM a partire dall'indirizzo decimale 64785. Per attivare il TIM da un programma BASIC si deve scrivere SYS(64785).

Con il Monitor si possono:

- leggere e/o modificare registri;
- leggere e/o modificare locazioni di memoria;
- far partire l'esecuzione di programmi;
- leggere e scrivere dati binari;
- ritornare il controllo al BASIC.

I comandi che TIM accetta sono:

M per verificare e correggere la memoria;
R per verificare e correggere registri;
G per iniziare l'esecuzione;
L per caricare in memoria;
S per memorizzare;
X per ritornare al BASIC.

Vediamo alcuni Esempi:

1. Il TIM attivato risponde con un punto. Per verificare e correggere da C000 a C010:

.M C0000,C010	scrive l'utente con indirizzi esadecimali
.:C000 1D C7 48 C6 35 CC EF C7	risponde il monitor dando i dati 8 per riga
.:C008 C5 CA DF CA 70 CF 23 CB	in esadecimale e completando anche l'ultimo
.:C010 9C C8 CC C8 34 67 70 CB	gruppo di 8 bytes.

Per modificare uno o più dei bytes evidenziati, si posiziona il cursore dello schermo nella giusta posizione e si corregge, premendo RETURN la correzione va in memoria.

2. Per evidenziare e/o modificare registri:

.R

	PC	IRQ	SR	AC	XR	YR	SP
.:	C6ED	E62E	00	20	00	F5	FE

Per modificare i registri procedere come in 1. Si tenga presente che prima di entrare nel Monitor i registri vengono salvati e poi ripristinati quando si esce dal Monitor.

3. Per iniziare l'esecuzione di un pezzo di programma:

. G indirizzo esadecimale — e l'esecuzione parte dall'indirizzo dato. L'indirizzo può anche essere omesso; in tale caso viene preso come indirizzo il contenuto del PC.

4. Per tornare al BASIC:

.X ed il sistema risponde con READY. Questo ritorno al BASIC non altera in alcun modo lo stato precedente della memoria.

5. Per caricare un programma:

.L "Nome programma", 01 — se 01 è il nome dell'apparecchiatura da cui leggere. Il programma viene caricato all'indirizzo nel quale si trova al momento del SAVE.

6. Per memorizzare un programma:

.S "Nome programma",01,0400,076D

dove 01 o altro è il numero dell'apparecchiatura e 0400 è l'indirizzo di inizio del programma da memorizzare e 076D è l'indirizzo di fine, tali indirizzi (quelli del momento ovviamente) sono necessari e devono essere in esadecimale.

Per cancellare un comando Monitor, si può usare RETURN, oppure per L,M ed S anche lo STOP. Se il Monitor è chiamato normalmente, appare il messaggio contenente il contatore del programma, la parola di stato, l'accumulatore, il registro indice X, il registro Y e il puntatore allo stack preceduto da C e asterisco, a indicare chiamata da CALL. Se viene eseguita dal programma un'istruzione BRK, essa provoca una interruzione con salvataggio nello stack del PC e del registro di stato e salto attraverso un vettore alle locazioni di indirizzo esadecimale 021B e 021C. Il TIM inizializza questo vettore per andare in onda in caso di BRK. In tale caso il messaggio iniziale del Monitor è preceduta da B e asterisco, a indicare la diversa chiamata.

Diamo un elenco di indirizzi iniziali di routine speciali, alle quali l'utente può saltare con l'istruzione JSR:

Indirizzi esadecimali	Azione
FFD2	stampa un carattere, nome simbolico WRT
FFCF	input di un carattere, nome simbolico RDT
FFE4	GET di un carattere, nome simbolico GET
FDD0	stampa un CR, nome simbolico CRLF
E775	stampa un byte, nome simbolico WROB
E7B6	legge un byte, nome simolico RDOB
E7E0	trasforma ASCII in esadecimale in A, nome simbolico HEXIT

Proponiamo un semplice programma per far comparire sul video i 64 caratteri ASCII. Il programma è:

Codice simbolico			Codice esadecimale		
CHSET	JSR	CRLF	20	D0	FD
	LDX	$20	A2	20	
LOOP	TXA		8A		
	JSR	WRT	20	D2	FF
	INX		E8		
	CPX	$60	E0	60	
	BNE	LOOP	D0	F7	
	BRK		00		
	JMP	CHSET	4C	3A	03

Per caricarlo con il Monitor nel buffer della seconda cassetta di indirizzo esadecimale 033A, procediamo così:

```
.M 033A,034B          variamo le posizioni di memoria come
                      scritto
.:033A 20 D0 FD A2 20 8A 20 D2
.:0342 FF E8 E0 60 D0 F7 00 4C
.:034A 3A 03 .............
```

poi scriviamo .G 033A e vedremo apparire sullo schermo i 64 caratteri ASCII seguiti da un messaggio B asterisco dovuto all'istruzione BRK.
Se vogliamo legare questo programma al BASIC, dobbiamo cambiare l'istruzione BRK in RTS che corrisponde a 60 in esadecimale. Fatto questo

cambiamento, possiamo scrivere nelle posizioni 1 e 2 con il Monitor l'indirizzo 033A. Possiamo ora uscire dal Monitor con .X e provare ad entrare in questo programma o con A = USR(0) o con SYS(826). Con l'uso del Monitor si caricano più facilmente i programmi in memoria perchè si possono usare i codici esadecimali. Il Monitor può anche essere chiamato con SYS(1024).

10.4. Lista del Monitor

```
CBM RESIDENT MONITOR......PAGE 0001

LINE # LOC    CODE       LINE

2514  FD11               ;COPYRIGHT 1978 BY
2515  FD11               ;COMMODORE INTERNATIONAL LIMITED

2517  FD11               NCMDS  =8
2518  FD11  A9 43        CALLE  LDA #'C          ;CALL ENTRY
2519  FD13  85 B5               STA TMPC
2520  FD15  D0 16               BNE B3
2521  FD17  A9 42        BRKE   LDA #'B          ;BREAK ENTRY
2522  FD19  85 B5               STA TMPC
2523  FD1B  D8                  CLD
2524  FD1C  4A                  LSR A            ;C SET FOR PC CORRECTION
2525  FD1D  68                  PLA
2526  FD1E  8D 05 02            STA YR           ;SAVE Y
2527  FD21  68                  PLA
2528  FD22  8D 04 02            STA XR           ;SAVE X
2529  FD25  68                  PLA
2530  FD26  8D 03 02            STA ACC          ;SAVE ACCUMULATOR
2531  FD29  68                  PLA
2532  FD2A  8D 02 02            STA FLGS         ;SAVE FLAGS
2533  FD2D  68           B3     PLA
2534  FD2E  69 FF               ADC #$FF         ;PC-1 FOR BREAK
2535  FD30  8D 01 02            STA PCL
2536  FD33  68                  PLA
2537  FD34  69 FF               ADC #$FF
2538  FD36  8D 00 02            STA PCH
2539  FD39  A5 90               LDA CINV         ;SAVE CUURENT IRQ VECTOR
2540  FD3B  8D 08 02            STA INVL
2541  FD3E  A5 91               LDA CINV+1
2542  FD40  8D 07 02            STA INVH
2543  FD43  BA                  TSX              ;SAVE CURRENT STACK POINTER
2544  FD44  8E 06 02            STX SP
2545  FD47  58                  CLI              ;CLEAR INTERRUPT DISABLE
2546  FD48  20 D0 FD     B5     JSR CRLF         ;PRINT ENTRY DATA
2547  FD4B  A6 B5               LDX TMPC         ;TYPE OF ENTRY (B OR C)
2548  FD4D  A9 2A               LDA #'*
2549  FD4F  20 84 E7            JSR WRTWO        ;WRITE '*C' OR '*B'
2550  FD52  A9 52               LDA #'R          ;DISPLAY REGISTERS COMMAND
2551  FD54  D0 1A               BNE S0           ;SKIP TO INTERPRET COMMAND

2553  FD56  A9 02        STRT   LDA #2           ;USER COMMAND INPUT
2554  FD58  85 77               STA TXTPTR       ;COMING FROM TEXT BUFFER
2555  FD5A  A9 00               LDA #0
2556  FD5C  85 DE               STA WRAP         ;ADDR WRAP AROUND FLAG
2557  FD5E  A2 0D               LDX #CR          ;START PROMPT WITH CRLF
2558  FD60  A9 2E               LDA #'.          ;A PROMPTING '.'
2559  FD62  20 84 E7            JSR WRTWO
2560  FD65  20 EB E7     ST1    JSR RDOC         ;INPUT COMMAND LINE
2561  FD68  C9 2E               CMP #'.          ;IGNORE PROMPTING '.'
2562  FD6A  F0 F9               BEQ ST1
2563  FD6C  C9 20               CMP #$20         ;SPAN BLANKS
2564  FD6E  F0 F5               BEQ ST1
```

```
LINE # LOC     CODE        LINE

2566  FD70  A2 07       S0     LDX #NCMDS-1     ;LOOKUP COMMAND
2567  FD72  DD E0 FD    S1     CMP CMDS,X
2568  FD75  D0 0B              BNE S2
2569  FD77  86 B4              STX SAVX         ;INDEX OF COMMAND IN TABLE
2570  FD79                     ;INDIRECT JMP FROM TABLE BY
2571  FD79                     ; PUSHING TARGET ADDRESS-1
2572  FD79                     ; THEN RTS
2573  FD79  BD E8 FD           LDA ADRH,X
2574  FD7C  48                 PHA
2575  FD7D  BD F0 FD           LDA ADRL,X
2576  FD80  48                 PHA
2577  FD81  60                 RTS
2578  FD82  CA          S2     DEX
2579  FD83  10 ED              BPL S1           ;LOOP FOR ALL COMMANDS

2581  FD85  6C FA 03           JMP (USRCMD)     ;ALLOW USER COMMANDS




2583  FD88  A5 FB       PUTP   LDA TMP0
2584  FD8A  8D 01 02           STA PCL
2585  FD8D  A5 FC              LDA TMP0+1
2586  FD8F  8D 00 02           STA PCH
2587  FD92  60                 RTS

2589  FD93                     ;DISPLAY MEM SUBR. SET AR=NUMBER
2590  FD93                     ;OF MEMORY BYTES DISPLAYED.
2591  FD93                     ;TMP0=ADR OF MEM DISPLAYED
2592  FD93                     ;
2593  FD93  85 B5       DM     STA TMPC
2594  FD95  A0 00              LDY #0
2595  FD97  20 CD FD    DM1    JSR SPACE        ;WR N BYTES
2596  FD9A  B1 FB              LDA (TMP0),Y     ;(TMP0)=ADR
2597  FD9C  20 75 E7           JSR WROB
2598  FD9F  20 D5 FD           JSR INCTMP
2599  FDA2  C6 B5              DEC TMPC
2600  FDA4  D0 F1              BNE DM1
2601  FDA6  60                 RTS
2602  FDA7                     ;READ AND STORE BYTE.
2603  FDA7                     ;NO STORE IF SPACE OR TMPC = 0.

2605  FDA7  20 B6 E7    BYTE   JSR RDOB
2606  FDAA  90 0D              BCC BY3          ;SPACE
2607  FDAC  A2 00              LDX #0           ;STORE BYTE
2608  FDAE  81 FB              STA (,MP0,X)
2609  FDB0  C1 FB              CMP (TMP0,X)     ;VERIFY WRITE
2610  FDB2  F0 05              BEQ BY3
2611  FDB4  68                 PLA              ;ERROR:CLEAR STACK
2612  FDB5  68                 PLA
2613  FDB6  4C F7 E7           JMP ERROPR
2614  FDB9  20 D5 FD    BY3    JSR INCTMP       ;INC TMP0 ADR
2615  FDBC  C6 B5              DEC TMPC
2616  FDBE  60                 RTS
```

```
LINE # LOC     CODE        LINE

2618  FDBF  A9 02       SETR   LDA #<FLGS       ;SET TO ACCESS REGS
2619  FDC1  85 FB              STA TMP0
2620  FDC3  A9 02              LDA #>FLGS
2621  FDC5  85 FC              STA TMP0+1
2622  FDC7  A9 05              LDA #5
2623  FDC9  60                 RTS

2625  FDCA  20 CD FD    SPAC2  JSR SPACE
2626  FDCD  A9 20       SPACE  LDA #$20
2627  FDCF  2C                 .BYT $2C
2628  FDD0  A9 0D       CRLF   LDA #$D
2629  FDD2  4C D2 FF           JMP $FFD2

2631  FDD5              ;INCREMENT (TMP0,TMP0+1) BY 1
2632  FDD5  E6 FB       INCTMP INC TMP0         ;LOW BYTE
2633  FDD7  D0 06              BNE SETWR
2634  FDD9  E6 FC              INC TMP0+1       ;HIGH BYTE
2635  FDDB  D0 02              BNE SETWR
2636  FDDD  E6 DE              INC WRAP
2637  FDDF  60          SETWR  RTS



2639  FDE0              ;COMMAND AND ADDRESS TABLE

2641  FDE0  3A          CMDS   .BYT ':'         ;MODIFY MEMORY
2642  FDE1  3B                 .BYT ';'         ;ALTER REGISTERS
2643  FDE2  52                 .BYT 'R'         ;DISPLAY REGS
2644  FDE3  4D                 .BYT 'M'         ;DISPLAY MEMORY
2645  FDE4  47                 .BYT 'G'         ;START EXECUTION
2646  FDE5  58                 .BYT 'X'         ;WARM START BASIC
2647  FDE6  4C                 .BYT 'L'         ;LOAD MEMORY
2648  FDE7  53                 .BYT 'S'         ;SAVE MEMORY
2649  FDE8  FE          ADRH   .BYT >ZZ1
2650  FDE9  FE                 .BYT >ZZ2
2651  FDEA  FE                 .BYT >ZZ3
2652  FDEB  FE                 .BYT >ZZ4
2653  FDEC  FE                 .BYT >ZZ5
2654  FDED  FF                 .BYT >ZZ6
2655  FDEE  FF                 .BYT >ZZ7
2656  FDEF  FF                 .BYT >ZZ8
2657  FDF0  88          ADRL   .BYT <ZZ1
2658  FDF1  96                 .BYT <ZZ2
2659  FDF2  22                 .BYT <ZZ3
2660  FDF3  57                 .BYT <ZZ4
2661  FDF4  CE                 .BYT <ZZ5
2662  FDF5  86                 .BYT <ZZ6
2663  FDF6  10                 .BYT <ZZ7
2664  FDF7  10                 .BYT <ZZ8
```

```
LINE # LOC     CODE        LINE

2666  FDF8  0D          REGK   .BYT CR,'    '
2666  FDF9  20 20
2667  FDFD  20 50              .BYT ' PC  IRQ  SR AC XR YR SP'

2669  FE15  98          ALTRIT TYA
2670  FE16  48                 PHA
2671  FE17  20 D0 FD           JSR CRLF
2672  FE1A  68                 PLA
2673  FE1B  A2 2E              LDX #'.
2674  FE1D  20 84 E7           JSR WRTWO
2675  FE20  4C CA FD           JMP SPAC2

2677  FE23  A2 00       DSPLYR LDX #0
2678  FE25  BD F8 FD    D2     LDA REGK,X
2679  FE28  20 D2 FF           JSR $FFD2
2680  FE2B  E8                 INX
2681  FE2C  E0 1D              CPX #29
2682  FE2E  D0 F5              BNE D2
2683  FE30  A0 3B              LDY #';
2684  FE32  20 15 FE           JSR ALTRIT
2685  FE35  AD 00 02           LDA PCH
2686  FE38  20 75 E7           JSR WROB
2687  FE3B  AD 01 02           LDA PCL
2688  FE3E  20 75 E7           JSR WROB
2689  FE41  20 CD FD           JSR SPACE
2690  FE44  AD 07 02           LDA INVH
2691  FE47  20 75 E7           JSR WROB
2692  FE4A  AD 08 02           LDA INVL
2693  FE4D  20 75 E7           JSR WROB
2694  FE50  20 8F FD           JSR SETR
2695  FE53  20 93 FD           JSR DM
2696  FE56  F0 39              BEQ BEQS1

2698  FE58  20 EB E7    DSPLYM JSR RDOC
2699  FE5B  20 A7 E7           JSR RDOA         ;READ START ADR
2700  FE5E  90 34              BCC ERRS1        ;ERR IF NO SA
2701  FE60  20 97 E7           JSR T2T2         ;SA TO TMP2
2702  FE63  20 EB E7           JSR RDOC         ;SKIP DELIMITER
2703  FE66  20 A7 E7           JSR RDOA         ;READ END ADR
2704  FE69  90 29              BCC ERRS1        ;ERR IF NO EA
2705  FE6B  20 97 E7           JSR T2T2         ;SA TO TMP0, EA TO TMP2

2707  FE6E  20 01 F3    DSP1   JSR STOP1        ;TEST FOR STOP KEY
2708  FE71  F0 1E              BEQ BEQS1
2709  FE73  A6 BE              LDX WRAP
2710  FE75  D0 1A              BNE BEQS1
2711  FE77  38                 SEC              ;DOUBLE BYTE COMPARE
2712  FE78  A5 FD              LDA TMP2
2713  FE7A  E5 FB              SBC TMP0
2714  FE7C  A5 FE              LDA TMP2+1
2715  FE7E  E5 FC              SBC TMP0+1
2716  FE80  90 0F              BCC BEQS1        ;EA LESS THAN SA
2717  FE82  A0 3A              LDY #':
2718  FE84  20 15 FE           JSR ALTRIT
2719  FE87  20 6A E7           JSR WROA
```

```
LINE # LOC     CODE       LINE

2720   FE8A   A9 08             LDA #8
2721   FE8C   20 93 FD          JSR DM           ;DISPLAY 8, INCR TMP0
2722   FE8F   F0 DB             BEQ DSP1

2724   FE91   4C 56 FD   BEQS1  JMP STRT

2726   FE94   4C F7 E7   ERRS1  JMP ERROPR

2728   FE97              ;ALTER REGISTERS

2730   FE97   20 B6 E7   ALTR   JSR RDOB         ;SKIP 2 SPACES
2731   FE9A   20 A7 E7          JSR RDOA         ;CY=0 IF SP
2732   FE9D   90 03             BCC AL2          ;SPACE
2733   FE9F   20 88 FD          JSR PUTP         ;ALTER PC
2734   FEA2   20 CF FF   AL2    JSR $FFCF
2735   FEA5   20 A7 E7          JSR RDOA
2736   FEA8   90 0A             BCC AL3
2737   FEAA   A5 FB             LDA TMP0
2738   FEAC   8D 08 02          STA INVL
2739   FEAF   A5 FC             LDA TMP0+1
2740   FEB1   8D 07 02          STA INVH
2741   FEB4   20 BF FD   AL3    JSR SETR         ;SET TO ALTER R'S
2742   FEB7   D0 0A             BNE A4

2744   FEB9              ;ALTER MEMORY - READ ADR AND DATA

2746   FEB9   20 B6 E7   ALTM   JSR RDOB         ;SKIP 2 SPACES
2747   FEBC   20 A7 E7          JSR RDOA         ;READ MEM ALTER ADR
2748   FEBF   90 D3             BCC ERRS1        ;CY=0, IF SPACE,ERR
2749   FEC1   A9 08             LDA #8           ;SET CNT = 8
2750   FEC3   85 B5      A4     STA TMPC
2751   FEC5   20 EB E7   A5     JSR RDOC
2752   FEC8   20 A7 FD          JSR BYTE
2753   FECB   D0 F8             BNE A5
2754   FECD   F0 C2      A9     BEQ BEQS1
2755   FECF   20 CF FF   GO     JSR $FFCF
2756   FED2   C9 0D             CMP #$0D         ;IF CR, EXIT
2757   FED4   F0 0C             BEQ G1
2758   FED6   C9 20             CMP #$20         ;IF NOT SPACE, ERR
2759   FED8   D0 BA             BNE ERRS1
2760   FEDA   20 A7 E7          JSR RDOA
2761   FEDD   90 03             BCC G1
2762   FEDF   20 88 FD          JSR PUTP
2763   FEE2   AE 06 02   G1     LDX SP
2764   FEE5   9A                TXS              ;ORIG OR NEW SP VALUE TO SP
2765   FEE6   78                SEI
2766   FEE7   AD 07 02          LDA INVH
2767   FEEA   85 91             STA CINV+1
2768   FEEC   AD 08 02          LDA INVL
2769   FEEF   85 90             STA CINV
2770   FEF1   AD 00 02          LDA PCH
2771   FEF4   48                PHA
2772   FEF5   AD 01 02          LDA PCL
2773   FEF8   48                PHA
2774   FEF9   AD 02 02          LDA FLGS
```

```
LINE # LOC     CODE        LINE

2775  FEFC  48                  PHA
2776  FEFD  AD 03 02            LDA ACC
2777  FF00  AE 04 02            LDX XR
2778  FF03  AC 05 02            LDY YR
2779  FF06  40                  RTI

2781  FF07  AE 06 02    EXIT    LDX SP
2782  FF0A  9A                  TXS
2783  FF0B  4C 89 C3            JMP READY        ;EXIT TO BASIC WARM STAR

2785  FF0E  4C F7 E7    ERRL    JMP ERROPR

2787  FF11              ZZZ1    =BUF+7

2789  FF11              ;MACHINE LANGUAGE LOAD ROUTINE

2791  FF11  A0 01       LD      LDY #1
2792  FF13  84 D4               STY FA           ;DEFAULT DEVICE #1
2793  FF15  88                  DEY
2794  FF16  84 D1               STY FNLEN
2795  FF18  84 9D               STY VERCK
2796  FF1A  A9 02               LDA #>ZZZ1       ;PLACE TO STORE NAME
2797  FF1C  85 DB               STA FNADR+1
2798  FF1E  A9 07               LDA #<ZZZ1
2799  FF20  85 DA               STA FNADR
2800  FF22  20 CF FF    L1      JSR $FFCF
2801  FF25  C9 20               CMP #'
2802  FF27  F0 F9               BEQ L1           ;SPAN BLANKS
2803  FF29  C9 0D               CMP #CR
2804  FF2B  F0 1A               BEQ L5           ;DEFAULT TO LOAD
2805  FF2D  C9 22               CMP #'"
2806  FF2F  D0 DD       L2      BNE ERRL         ;FILE NAME MUST BE NEXT
2807  FF31  20 CF FF    L3      JSR $FFCF
2808  FF34  C9 22               CMP #'"
2809  FF36  F0 24               BEQ L8           ;END OF NAME
2810  FF38  C9 0D               CMP #CR          ;DEFAULT A LOAD
2811  FF3A  F0 0B               BEQ L5
2812  FF3C  91 DA               STA (FNADR)Y
2813  FF3E  E6 D1               INC FNLEN
2814  FF40  C8                  INY
2815  FF41  C0 10               CPY #16
2816  FF43  F0 C9       L4      BEQ ERRL         ;FILE NAME TOO LONG
2817  FF45  D0 EA               BNE L3
2818  FF47  A5 B4       L5      LDA SAVX
2819  FF49  C9 06               CMP #6
2820  FF4B  D0 E2       L6      BNE L2           ;NOT A LOAD
2821  FF4D  20 22 F3            JSR LD15
2822  FF50  20 E6 F0            JSR TWAIT
2823  FF53  A5 96               LDA SATUS
2824  FF55  29 10               AND #SPERR
2825  FF57  D0 F2       L7      BNE L6           ;LOAD ERROR
2826  FF59  4C 56 FD            JMP STRT
2827  FF5C  20 CF FF    L8      JSR $FFCF
2828  FF5F  C9 0D               CMP #CR
2829  FF61  F0 E4               BEQ L5           ;DEFUALT LOAD
```

```
LINE # LOC      CODE         LINE

2830   FF63  C9 2C                  CMP #',
2831   FF65  D0 F0        L9        BNE L7          ;BAD SYNTAX
2832   FF67  20 86 E7               JSR RDOB
2833   FF6A  29 0F                  AND #$F
2834   FF6C  F0 D5        L10       BEQ L4          ;DEVICE 0
2835   FF6E  C9 03                  CMP #3
2836   FF70  F0 FA        L11       BEQ L10         ;DEVICE 3
2837   FF72  85 D4                  STA FA
2838   FF74  20 CF FF               JSR $FFCF
2839   FF77  C9 0D                  CMP #CR
2840   FF79  F0 CC                  BEQ L5          ;DEFAULT LOAD
2841   FF7B  C9 2C                  CMP #',
2842   FF7D  D0 E6        L12       BNE L9          ;BAD SYNTAX
2843   FF7F  20 A7 E7               JSR RDOA
2844   FF82  20 97 E7               JSR T2T2
2845   FF85  20 CF FF               JSR $FFCF
2846   FF88  C9 2C                  CMP #',
2847   FF8A  D0 F1        L13       BNE L12         ;MISSING END ADDR
2848   FF8C  20 A7 E7               JSR RDOA
2849   FF8F  A5 FB                  LDA TMP0
2850   FF91  85 C9                  STA EAL
2851   FF93  A5 FC                  LDA TMP0+1
2852   FF95  85 CA                  STA EAH
2853   FF97  20 97 E7               JSR T2T2
2854   FF9A  20 CF FF     L20       JSR $FFCF
2855   FF9D  C9 20                  CMP #$20
2856   FF9F  F0 F9                  BEQ L20
2857   FFA1  C9 0D                  CMP #CR
2858   FFA3  D0 E5        L14       BNE L13         ;MISSING CR AT END
2859   FFA5  A5 B4                  LDA SAVX
2860   FFA7  C9 07                  CMP #7
2861   FFA9  D0 F8                  BNE L14
2862   FFAB  20 A4 F6               JSR SV5
2863   FFAE  4C 56 FD               JMP STRT

2865   FFB1               ZZ1=ALTM-1
2866   FFB1               ZZ2=ALTR-1
2867   FFB1               ZZ3=DSPLYR-1
2868   FFB1               ZZ4=DSPLYM-1
2869   FFB1               ZZ5=GO-1
2870   FFB1               ZZ6=EXIT-1
2871   FFB1               ZZ7=LD-1
2872   FFB1               ZZ8=LD-1
```

APPENDICE A

MESSAGGI DI ERRORE

Quando c'è un errore il PET si rimette in attesa di un comando esterno ed interrompe l'esecuzione del programma. Sullo schermo viene visualizzato il messaggio di errore; tale messaggio termina con ERROR IN XX, dove XX è il numero della linea che ha causato l'errore, se stava lavorando sotto controllo di un programma memorizzato; mentre termina con ERROR solamente se stava eseguendo frasi BASIC dirette o comandi.

Messaggio	Significato
CAN'T CONTINUE	Si è tentato di dare CONT, ma la situazione è una di queste: - non c'è un programma in memoria, - si è appena avuto un qualunque errore, - si è appena scritta o corretta una linea. Se necessario correggete l'eventuale errore e per proseguire date o il RUN o un GOTO xx.
BAD SUBSCRIPT	Potete: - o avere riferito indici maggiori di quelli richiesti con la DIM, - o avete dimensionato una variabile con un indice e la richiamate con due indici, cioè l'uso della variabile con indici non concorda con la sua definizione.
DIVISION BY ZERO	Avete tentato di dividere per zero e questo non è possibile. Correggete la frase di calcolo.
ILLEGAL DIRECT	Avete usato frasi tipo INPUT, GET o DEF in modo diretto. Non è consentito.
ILLEGAL QUANTITY	Si ha questo errore quando si tenta di usare una funzione con un parametro fuori dai limiti consentiti. Può essere dovuto a diversi motivi: - indici di una variabile con indici, fuori dall'intervallo 1-32767

Messaggio	Significato
	- argomento del LOG negativo o nullo, - argomento di SQR negativo, - A↑B con A = 0 e/o B non intero, - chiamata a USR prima che l'indirizzo della subroutine in linguaggio macchina sia stato sistemato, - uso di una funzione di stringa con indici fuori dall'intervallo 1-255, - ON ... GOTO fuori dall'intervallo, - indirizzo specificato superiore a 65535, - parametro di WAIT, POKE o TAB o SPC fuori dall'intervallo 0-255.
NEXT WITHOUT FOR	Si ha un NEXT che non è preceduto dal relativo FOR.
OUT OF DATA	Si è usato il READ per leggere più dati di quanti sono contenuti nel blocco dati.
OVERFLOW	Il risultato è troppo grande ed il BASIC non lo può mantenere, si ha se un risultato è maggiore di 1.70141184 E + 38. Se un risultato è troppo piccolo non si ha errore, ma se esso è minore di 2.93873587 E-39 si ha zero.
REDIM'D ARRAY	È stato dato due volte il dimensionamento di una variabile con indice, oppure si è riferita dapprima con dimensionamento implicito e poi si dà una DIM.
RETURN WITHOUT GOSUB	È stato incontrato un RETURN, ma non c'è stato un precedente GOSUB.
STRING TOO LONG	L'utente ha tentato di concatenare due stringhe la cui lunghezza complessiva supera 255 caratteri.
STRING FORMULA TOO COMPLEX	L'espressione era troppo complessa; conviene spezzare in due espressioni più semplici.
SYNTAX	Nella frase c'è un errore di sintassi; può essere: - parentesi non bilanciate, - caratteri illegali, - punteggiatura scorretta, ecc.

Messaggio	Significato
TYPE MISMATCH	Errore nel tipo di dati, può essere: - variabile numerica = variabile stringa o viceversa, - una funzione prevedeva un argomento stringa ed ha ricevuto un numero.
UNDEF'D STATEMENT	GOTO, GOSUB o THEN con numero di riferimento a linea inesistente.
UNDEF'D FUNCTION	Riferimento ad una funzione utente non ancora definita.
BAD DATA	In operazioni file si trasmettono o ricevono dati di tipo errato.
FILE OPEN	Tentativo di aprire un file già aperto.
FILE NOT OPEN	Tentativo di operare sul file non aperto.
LOAD	Nel caricare un programma dalle cassette magnetiche ci sono errori di registrazione.
NOT INPUT FILE	Avete cercato di usare per INPUT un file aperto per OUTPUT.
NOT OUTPUT FILE	Avete cercato di usare PRINT per un file aperto per INPUT.
DEVICE NOT PRESENT	Avete tentato di usare un dispositivo non presente.
VERIFY	Il contenuto della memoria e del file che state verificando non concordano.
FILE NOT FOUND	Il nome del file non si trova sulla device.
REDO FROM START	A una richiesta di input da tastiera si risponde con dati di tipo errato.
??	A una richiesta di input da tastiera si risponde con meno dati di quelli richiesti.
OUT OF MEMORY	Può essere dovuto a diverse cause.

Nota: è possibile si verifichi un errore di questo tipo: avete scritto un programma con un ciclo di INPUT senza possibilità di uscita:

```
10 INPUT A,B
20 PRINT A,B
30 GOTO 10
```

Da questo ciclo di istruzioni non riuscite più ad uscire perchè quando il PET è in attesa di input non sente il comando STOP da tastiera. L'unica procedura possibile è spegnere il PET e riaccenderlo oppure dare un dato di tipo sbagliato. Vi raccomandiamo allora di programmare così per esempio:

```
10 INPUT A,B
20 PRINT A,B
30 IF A = 9999 GOTO 50
40 GOTO 10
50 STOP
```

In tale modo uscite dal ciclo scrivendo un numero particolare per A.

APPENDICE B

ELENCO ERRORI DISCO RICEVUTI ATTRAVERSO IL CANALE 15

Il formato dei messaggi è il seguente:

N. messaggio	MESSAGGIO	TRACCIA	SETTORE
		o 00	o 00

Elenchiamo i messaggi dividendoli per tipo e dando un breve commento.

Messaggi di stato

00 OK 00 00

nessun errore

01 FILES SCRATCHED # FILES 00

il FILES è stato cancellato

Messaggi per errori lettura

20 READ ERROR T S

non è stata trovata l'intestazione del blocco. Il controller del disco non riesce ad individuare l'intestazione del blocco dati richiesto. Può esserci un settore errato oppure l'intestazione è andata persa.

21 READ ERROR T S

manca il carattere di sincronizzazione. Il controller non riesce ad individuare il carattere di sincronizzazione nella traccia desiderata. È causato da un cattivo allineamento della testina di lettura, dalla mancanza del disco o da un errore hardware.

22 READ ERROR T S

blocco dati assente. È stato richiesto di leggere un blocco dati che non è stato precedentemente scritto bene. Avviene in corrispondenza di comandi BLOCK ed indica una richiesta illegale di traccia o settore.

23 READ ERROR T S

errore di checksum in un blocco di dati. Ci può essere un errore in qualche bytes del blocco dati. Dopo aver letto il blocco la somma di controllo è errata. Può anche dipendere da problemi di messa a terra.

23 READ ERROR T S

errore di decodifica di byte. I dati o l'intestazione di un file sono stati letti nella memoria DOS, ma si è generato un errore hardware a causa di una errata configurazione di bit in qualche byte. Può indicare problemi di messa a terra.

24 READ ERROR T S

errore di checksum nell'intestazione. È stato trovato un errore nell'intestazione del blocco dati richiesto. Il blocco non viene letto nella memoria DOS. Può indicare anche problemi di messa a terra.

Messaggi per errori di scrittura

25 WRITE ERROR T S

errore di verifica di scrittura. Non si trova corrispondenza tra quanto scritto e quanto contenuto nella memoria DOS.

26 WRITE PROTECT ON T S

si tenta di scrivere su un dischetto nel quale è schermata la finestra per avere protezione di scrittura.

28 WRITE ERROR T S

blocco di dati lungo. Il controller dopo aver scritto un blocco cerca di individuare il carattere di sincronizzazione del prossimo blocco e non lo trova entro un certo tempo. Può dipendere da una cattiva formattazione del dischetto o da un guasto.

29 DISK IS MISMATCH T S

dischetto non inizializzato o con intestazione errata.

Errori di sintassi

30 SYNTAX ERROR 00 00

errore di tipo generico. Il DOS non può interpretare il comando. Il comando è probabilmente scritto male.

31 SYNTAX ERROR 00 00

comando invalido. Il DOS non riconosce il comando. Il comando deve iniziare nella prima posizione della linea.

32 SYNTAX ERROR 00 00

linea lunga. Il comando è più lungo di 40 caratteri.

33 SYNTAX ERROR 00 00

nome di file illegale. Il nome file usato in LOAD o SAVE è invalido.

34 SYNTAX ERROR 00 00

file non riconoscibile. Probabilmente c'è un errore nella stringa comando circa il nome del file.

Errori di file

60 WRITE FILE OPEN 00 00
si apre per lettura un file che non era stato chiuso.

61 FILE NOT OPEN 00 00
si tenta di utilizzare un file non aperto. In certi casi non si ha questo messaggio ed il comando viene ignorato.

62 FILE NOT FOUND 00 00
il file richiesto non si trova sul disco indicato.

63 FILE EXISTS 00 00
esiste già un file dello stesso nome.

64 FILE TIPE MISMATCH 00 00
il tipo di file specificato non corrisponde a quello descritto nella DIRECTORY del disco.

65 NO BLOCK T S
tale messaggio si presenta in corrispondenza al comando B-A. Sta ad indicare che il blocco da allocare è già stato allocato in precedenza. I parametri T ed S se diversi da zero indicano la traccia ed il settore disponibili con numero più alto di quello richiesto. Se i parametri sono 0 non è disponibile alcun blocco.

Errori di sistema

70 NO CHANNEL 00 00
nessun canale è disponibile. Il canale richiesto non è disponibile, oppure i canali sono tutti impegnati. Nel DOS possono essere aperti contemporaneamente al massimo 5 files SEQUENZIALI o 6 files ad ACCESSO DIRETTO.

71 DIR ERROR 00 00
errore nella DIRECTORY. La BAM non può essere accoppiata con il conto interno. L'errore si ha per problemi di allocazione del BAM o se si è scritto sul BAM nella memoria DOS. Per superare tale problema è necessario riinizializzare il disco; in tale modo però vengono disabilitati i files attivi.

72 DISK FULL 00 00
il disco è stato completamente usato oppure la DIRECTORY è completamente piena. Nella DIRECTORY si possono avere 152 registrazioni.

APPENDICE C

UTILIZZO DELLA MEMORIA

Mappa della memoria in blocchi di 4K			
N. Blocco	**Tipo**	**Indirizzo inizio esadecimale**	**Funzione**
0	RAM	0000	Area lavoro Sistema Operativo e Interprete BASIC e programma utente
1	RAM	1000	Programma utente e variabili
2	---	2000	Espansione RAM
3	---	3000	Espansione RAM
4	---	4000	Espansione RAM
5	---	5000	Espansione RAM
6	---	6000	Espansione RAM
7	---	7000	Espansione RAM
8	RAM	8000	Memoria per lo schermo
9	---	9000	Espansione ROM
10	---	A000	Espansione ROM
11	---	B000	Espansione ROM
12	ROM	C000	Prima parte Interprete BASIC
13	ROM	D000	Seconda parte Interprete BASIC
14	ROM	E000	Screen editor (Programma gestione schermo)
	I/O	E800	Trattamento I/O
15	ROM	F000	Sistema operativo

Mappa del blocco zero in pagine di 256 bytes		
Pagina	**Indirizzo inizio esadecimale**	**Funzione**
0	0000	Area lavoro BASIC
1	0100	Area Stack
2	0200	Area di lavoro Sistema Operativo
3	0300	Buffers per cassette magnetiche
4 ÷ 15	0400	Area utente per programmi

Mappa del blocco 14 in segmenti di 2K			
Pagina	**Tipo**	**Indirizzo inizio esadecimale**	**Funzione**
0	ROM	E000	Programma SCREEN EDITOR
1	I/O	E800	I/O del calcolatore

Indirizzi base delle periferiche di I/O (nella pagina 1 del blocco 14)			
Pagina	**Tipo**	**Indirizzo inizio esadecimale**	**Funzione**
0	PIA	E810	Tastiera (KEYBO-ARD)
1	PIA	E820	IEEE-488
2	VIA	E840	USR PORT per casset-te

Mappa della pagina zero della memoria RAM		
Indirizzi		**Descrizione**
DA	A	
000		4C costante, cioè istruzione JMP del microprocessore 6502
001	002	indirizzo della funzione USR
003		delimitatore di partenza
004		delimitatore di fine
005		contatore generale del BASIC
Valutazione delle variabili		
6		flag per ricordare le variabili dimensionate
7		flag per il tipo delle variabili; 0 = numeriche; 1 = stringhe
8		flag per variabili intere
9		flag per le parole riservate
10		flag che permette di usare gli indici
11		flag per INPUT o READ
12		flag per segno di TAN
13		flag per sopprimere OUTPUT (+ normale, — soppresso)
14		numero del canale di I/O attivo
15		non usato
16		non usato
17	18	numero della linea
19		indice per il prossimo descrittore disponibile
20	21	puntatore all'ultima stringa temporanea
22	29	tabella dei descrittore a due bytes che puntano alle variabili
30	31	indice indiretto 1
32	33	indice indiretto 2
34	39	pseudo registro per gli operandi delle funzioni
Controllo della memoria		
40	41	puntatore all'inizio dell'area programma BASIC
42	43	puntatore all'inizio dell'area variabili
44	45	puntatore alla tabella delle matrici (variabili con indice)
46	47	puntatore alla fine delle variabili

Indirizzi		Descrizione
DA	A	
48	49	puntatore all'inizio dell'area per le stringhe
50	51	puntatore alla fine dell'area per le sringhe
52	53	indirizzo più alto della RAM
54	55	numero della linea in esecuzione; un 2 nel byte 54 significa esecuzione comando in modo diretto
56	57	numero di linea per il comando CONTINUE
58	59	puntatore alla prossima linea da eseguire
60	61	numero della linea per messaggio errore
62	63	puntatore per la frase DATA
Valutazione delle espressioni		
64	65	provenienza dell'INPUT
66	67	nome della variabile in uso
68	69	puntatore alla variabile in memoria
70	71	puntatore alla variabile del ciclo FOR-NEXT
72	73	puntatore nella tabella all'operatore in uso
74		maschera speciale per l'operatore in uso
75	76	puntatore alla definizione della funzione
77	78	puntatore alla descrizione della stringa
79		lunghezza della stringa
80		costante usata dalla routine FRE
81		4C costante, cioè istruzione JMP
82	83	vettore per risolvere le funzioni
84	89	accumulatore N. 3 per operazioni floating-point (FLAG 3)
90	91	puntatore 1 per trasferimento blocchi
92	93	puntatore 2 per trasferimento blocchi
94	99	accumulatore N. 1 per operazioni floating-point (usato per la funzione USR) (FAC 1)
100		duplicato del segno della mantissa di FAC 1
101		contatore del numero degli shift necessari per normalizzare FAC 1
102	107	accumulatore N. 2 per operazioni floating-point
108		byte di overflow per argomenti floating
109		duplicato del segno della mantissa
110	111	puntatore per le conversioni in FAC

Indirizzi		Descrizione
DA	A	
Sottoprogrammi RAM		
112		prende il prossimo carattere dal testo BASIC
118		riprende il carattere corrente dal testo BASIC
119	120	puntatore al testo sorgente
136	140	prossimo numero random in memoria
Sistema operativo		
141	143	orologio per 24 ore (intervallo 1/60 sec)
144	145	vettore IRQ
146	147	vettore BRK
148	149	vettore NMI
150		byte di stato per I/O
151		indice ultima chiave
152		tasto shift
153	154	fattore di correzione per orologio
157		flag di verifica
158		indice alla coda per la tastiera
159		indicatore di campo inverso
160	166	diversi usi
167		flag per cursore on
168		contatore istanti per tremolio cursore
169		carattere del cursore
170		salvataggio carattere durante tremolio
171	173	diversi usi
174		puntatore nella tabella dei files logici
175		numero per difetto (in mancanza) del numero della periferica di input
176		numero per difetto (in mancanza) del numero della periferica di output
177		parità verticale per nastro
178	185	usi diversi
186		SYNC nel conto dell'intestazione del nastro
187	188	puntatore alla cassetta attiva
189		usi diversi
190		per errore nastro
191		blocco corto in lettura
192		indice all'indirizzo per la correzione di errori nastro

Indirizzi		Descrizione
DA	A	
193		usi diversi
194		flag per cassette
195		contatore per i secondi prima di scrivere i dati
196	197	puntatore alla posizione del cursore
198		
199	200	indirizzo di partenza per LOAD
201	202	indirizzo di fine per LOAD
203		
204		
205		flag per gli apici
206	208	usi diversi
209		lunghezza del nome del file in uso
210		indirizzo logico del file in uso
211		indirizzo secondario file in uso
212		indirizzo primario del file in uso
213	217	usi diversi
218	219	indirizzo del nome del file in uso
220	221	usi diversi
222		contatore per blocco cassetta in lettura
223		usi diversi
224	248	tabella degli LBB degli indirizzi di partenza delle linee del video
249		
250		
251		
252		
253	254	

PAGINA UNO DELLA MEMORIA RAM

Gli ultimi 62 bytes sono usati per correggere gli errori in lettura dei nastri. Ci sono poi i buffers necessari per trasformare il contenuto dei FAC in caratteri stampabili. Il resto della pagina è usato per memorizzare i ritorni per i GOSUB, il contesto dei cicli FOR/NEXT e per l'area di lavoro Stack.

Mappa della pagina due della memoria RAM		
Indirizzi		**Descrizione**
DA	A	
256	511	buffers di input per il BASIC
512	513	contatore del programma
514		stato del processore
515		accumulatore
516		indice x
517		indice y
518		puntatore allo Stack
519	520	IRQ modificabile dall'utente
593	602	numero logico file
603	612	numero della periferica principale
613	622	indirizzo secondario
623	633	buffer della tastiera
634	825	buffer per cassetta 1
826	1017	buffer per cassetta 2
1018	1019	non usate

MEMORIZZAZIONE DELLE VARIABILI IN BASIC

Viene assegnato spazio alle variabili al momento della loro definizione, e cioè:

- per le variabili singole, quando compaiono a sinistra di un uguale o in una lista di INPUT o READ,
- per le variabili con indice, quando si ha la DIM, a meno che, questa sia omessa ed allora il sistema assegna 10 per ogni indice.

Per ogni variabile singola sono usati 7 bytes, sia essa una stringa, un numero

o una funzione definita dall'utente. I primi due bytes danno il nome della variabile come dal seguente schema:

	byte 1	byte 2
INTERI	primo carattere + 128	secondo carattere + 128 o solo 128
FLOATING	primo carattere	secondo carattere o zero
STRINGHE	primo carattere	secondo carattere + 128 o solo 128

Gli altri 5 bytes danno il valore della variabile o la descrizione dei dati ai quali la variabile si riferisce, secondo il seguente schema:

<table>
<tr><th></th><th>byte 3</th><th>byte 4</th><th>byte 5</th><th>byte 6</th><th>byte 7</th></tr>
<tr><td rowspan="2">INTERI</td><td colspan="2">valore attuale</td><td rowspan="2">0</td><td rowspan="2">0</td><td rowspan="2">0</td></tr>
<tr><td>256 * HI</td><td>LO</td></tr>
<tr><td rowspan="2">FLOATING</td><td colspan="5">valore attuale del numero binario floating point</td></tr>
<tr><td>esponente</td><td colspan="4">mantissa</td></tr>
<tr><td rowspan="2">STRINGHE</td><td rowspan="2">contatore
caratteri</td><td colspan="2">puntatore</td><td colspan="2"></td></tr>
<tr><td>LO</td><td>HI</td><td>0</td><td>0</td></tr>
</table>

NOTA: HI significa byte di valore più significativo
LO significa byte di valore meno significativo.

La variabile stringa punta ad una locazione della memoria di indirizzo alto, dove la stringa è memorizzata. Le locazioni 124 e 125 contengono l'indirizzo dove inizia il nome di una variabile singola. Tale indirizzo viene incrementato di 7 per accedere al nome di ogni variabile che viene aggiunta. Le locazioni 126 e 127 contengono l'indirizzo della fine delle variabili singole e quindi dell'inizio delle variabili con indice. La struttura di memorizzazione delle variabili con indice si compone di due parti: la testata e gli elementi. La testata è di 7 bytes per una variabile avente una sola dimensione, di 9 bytes per una variabile avente due dimensioni, e così via aumentando di 2 bytes per ogni dimensione in più. Nello schema seguente è riportata la testata.

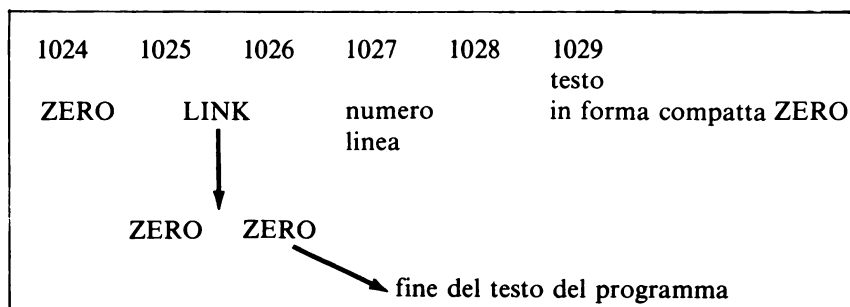
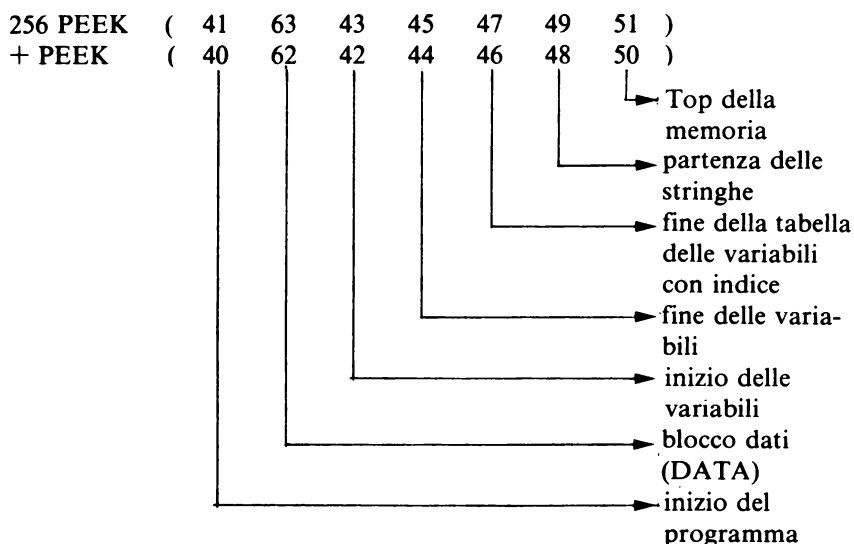
Testata della variabile con indice				
byte 1 byte 2	byte 3 byte 4	byte 5	byte 6 byte 7	byte 8 byte 9
come per le var. singole per distinguere il tipo	numero totale bytes occupati e quindi puntatore alla prossima variabile	numero dimensioni	numero di elementi dell'ultima dimensione	numero elementi penultima dimensione

Gli elementi della variabile seguono la testata, ma occupano meno spazio delle variabili singole; infatti ci vogliono:

- 2 bytes per ogni numero intero;
- 5 bytes per ogni numero floating point;
- 3 bytes per ogni stringa (contatore caratteri e puntatore).

Si ricorda che i numeri negativi sono memorizzati nella forma del complemento a 2. Le locazioni 128 e 129 contengono l'indirizzo dove termina la tabella delle variabili con indice.

MEMORIZZAZIONE DELLE FRASI BASIC

Il programma inizia all'indirizzo 1024; tale byte contiene sempre zero. I due bytes seguenti, 1025 e 1026 contengono il LINK e cioè l'indirizzo del LINK della frase seguente di programma. I due bytes seguenti contengono il numero della linea (che può andare da 1 a 63999). Segue la frase BASIC in forma compatta; la frase BASIC termina con un byte a zero. La fine del programma è segnalata dai due bytes di LINK a zero.

PRINCIPALI PUNTATORI NELLA MEMORIA RAM

CONSIGLI PRATICI

Da quanto detto si può dedurre che si vuole avere un programma veloce, si devono definire tutte le variabili singole, assegnando loro un valore, prima di definire ed inizializzare le variabili con indice. Se non si opera così, ogni volta che si definisce una nuova variabile singola, il sistema deve operare degli spostamenti sulle variabili con indice. Se si vuole risparmiare spazio in memoria è bene ricordare questi principi:

1. usare linee di programma con più di una frase BASIC, separando le istruzioni con i due punti;
2. non mettere spazi inutili nelle frasi di programma;
3. usare pochi REM e riportare a parte la documentazione del programma;
4. usare variabili invece di costanti, cioè definire le costanti con un nome e poi richiamarle con quel nome;
5. non usare END alla fine del programma;
6. riutilizzare le stesse variabili in punti diversi del programma, se possibile;
7. fare uso della tecnica dei sottoprogrammi e servirsi dei GOSUB;
8. usare le posizioni zero delle variabili con indice.

Un modo per accrescere la velocità del programma è quello di predisporre le variabili in modo che si trovino per prime quelle che vengono usate di più nel programma. Un'altro modo è di usare i NEXT senza variabile per chiudere i FOR.

APPENDICE D

CARATTERI DEL PET

Nelle pagine seguenti è riportata una tabella completa dei caratteri dei calcolatori della COMMODORE; diamo qui alcune note esplicative.
La colonna denominata PET si riferisce ad un calcolatore con tastiera grande. La colonna PET ASCII riporta il numero decimale ed esadecimale (HEX) corrispondente al carattere e che può essere usato con le due funzioni: ASC ( ) e CHR$( ).

La colonna PEEK/POKE riporta i valori decimali che si leggono o si scrivono con le rispettive funzioni nei bytes della memoria per ottenere i caratteri. Molti caratteri compaiono più di una volta, perchè hanno diversi valori PET ASCII corrispondenti. Nella colonna "Caratteri Standard" sono riportati i caratteri di cui il calcolatore normalmente dispone quando viene acceso. Per la tastiera grafica (colonna PET) questa situazione corrisponde al valore 12 nel byte 59468. Per la tastiera CBM questa situazione corrisponde invece al valore 14 nel byte 59468. Quando la colonna CBM non riporta un carattere, questo significa che esso è uguale a quello della colonna PET. Nella colonna "Caratteri Alternativi" sono riportati i caratteri di cui il calcolatore dispone dopo aver modificato il valore del byte 59468. Per la tastiera grafica i caratteri si ottengono scrivendo POKE 59468,14. Per la tastiera CBM i caratteri si ottengono scrivendo POKE 59468,12. Anche qui sono riportati solo i caratteri che differiscono da quelli della colonna PET. Come si può vedere i caratteri dell'ultima parte della tabella non hanno valori nella colonna PET ASCII, questo significa che per ottenerli non si possono usare le funzioni ASC e CHR$, ma solo leggerli con PEEK e scriverli con POKE.

Tabella D-1. Caratteri del sistema PET/CBM (continua)

Caratteri Standard PET	Caratteri Standard CBM	Caratteri Alternativi PET	Caratteri Alternativi CBM	PET ASCII DEC	PET ASCII HEX	PEEK/POKE
				0	00	
				1	01	
				2	02	
STOP		STOP		3	03	
				4	04	
				5	05	
				6	06	
				7	07	
				8	08	
				9	09	
				10	0A	
				11	0B	
				12	0C	
RETURN		RETURN		13	0D	
				14	0E	
				15	0F	
				16	10	
CRSR↓		CRSR↓		17	11	
RVS		RVS		18	12	
HOME		HOME		19	13	
DELETE		DELETE		20	14	
				21	15	
				22	16	
				23	17	
				24	18	
				25	19	
				26	1A	
				27	1B	
				28	1C	
CRSR→		CRSR→		29	1D	
				30	1E	
				31	1F	
␣	␣	␣	␣	32	20	32
!		!		33	21	33
"		"		34	22	34
#		#		35	23	35
$		$		36	24	36
%		%		37	25	37
&		&		38	26	38
'		'		39	27	39
(		(		40	28	40
)		)		41	29	41
*		*		42	2A	42
+		+		43	2B	43
,		,		44	2C	44
-		-		45	2D	45
.		.		46	2E	46
/		/		47	2F	47
0		0		48	30	48
1		1		49	31	49
2		2		50	32	50
3		3		51	33	51
4		4		52	34	52
5		5		53	35	53
6		6		54	36	54
7		7		55	37	55
8		8		56	38	56
9		9		57	39	57
				58	3A	58
;		;		59	3B	59
<		<		60	3C	60
=		=		61	3D	61
>		>		62	3E	62
?		?		63	3F	63
@		@		64	40	0

Caratteri Standard PET	Caratteri Standard CBM	Caratteri Alternativi PET	Caratteri Alternativi CBM	PET ASCII DEC	PET ASCII HEX	PEEK/POKE
A	a	a	A	65	41	1
B	b	b	B	66	42	2
C	c	c	C	67	43	3
D	d	d	D	68	44	4
E	e	e	E	69	45	5
F	f	f	F	70	46	6
G	g	g	G	71	47	7
H	h	h	H	72	48	8
I	i	i	I	73	49	9
J	j	j	J	74	4A	10
K	k	k	K	75	4B	11
L	l	l	L	76	4C	12
M	m	m	M	77	4D	13
N	n	n	N	78	4E	14
O	o	o	O	79	4F	15
P	p	p	P	80	50	16
Q	q	q	Q	81	51	17
R	r	r	R	82	52	18
S	s	s	S	83	53	19
T	t	t	T	84	54	20
U	u	u	U	85	55	21
V	v	v	V	86	56	22
W	w	w	W	87	57	23
X	x	x	X	88	58	24
Y	y	y	Y	89	59	25
Z	z	z	Z	90	5A	26
[	[	[	[	91	5B	27
\	\	\	\	92	5C	28
]	]	]	]	93	5D	29
↑	↑	↑	↑	94	5E	30
←	←	←	←	95	5F	31
				96	60	32
!	!	!	!	97	61	33
"	"	"	"	98	62	34
#	#	#	#	99	63	35
$	$	$	$	100	64	36
%	%	%	%	101	65	37
&	&	&	&	102	66	38
				103	67	39
(	(	(	(	104	68	40
)	)	)	)	105	69	41
*	*	*	*	106	6A	42
+	+	+	+	107	6B	43
,	,	,	,	108	6C	44
-	-	-	-	109	6D	45
.	.	.	.	110	6E	46
/	/	/	/	111	6F	47
0	0	0	0	112	70	48
1	1	1	1	113	71	49
2	2	2	2	114	72	50
3	3	3	3	115	73	51
4	4	4	4	116	74	52
5	5	5	5	117	75	53
6	6	6	6	118	76	54
7	7	7	7	119	77	55
8	8	8	8	120	78	56
9	9	9	9	121	79	57
				122	7A	58
;	;	;	;	123	7B	59
<	<	<	<	124	7C	60
=	=	=	=	125	7D	61
>	>	>	>	126	7E	62
?	?	?	?	127	7F	63
				128	80	64

Tabella D-2. Caratteri del sistema PET/CBM (continua)

Caratteri Standard PET	Caratteri Standard CBM	Caratteri Alternativi PET	Caratteri Alternativi CBM	PET ASCII DEC	PET ASCII HEX	PEEK/POKE
				129	81	65
				130	82	66
RUN		RUN		131	83	67
				132	84	68
				133	85	69
				134	86	70
				135	87	71
				136	88	72
				137	89	73
				138	8A	74
				139	8B	75
				140	8C	76
Shifted RETURN		Shifted RETURN		141	8D	77
				142	8E	78
				143	8F	79
				144	90	80
CRSR↑		CRSR↑		145	91	81
RVS Off		RVS Off		146	92	82
CLR Screen		CLR Screen		147	93	83
INSERT		INSERT		148	94	84
				149	95	85
				150	96	86
				151	97	87
				152	98	88
				153	99	89
				154	9A	90
				155	9B	91
				156	9C	92
CRSR←		CRSR←		157	9D	93
				158	9E	94
				159	9F	95
Shifted ␢		Shifted ␢		160	A0	96
				161	A1	97
				162	A2	98
				163	A3	99
				164	A4	100
				165	A5	101
				166	A6	102
				167	A7	103
				168	A8	104
				169	A9	105
				170	AA	106
				171	AB	107
				172	AC	108
				173	AD	109
				174	AE	110
				175	AF	111
				176	B0	112
				177	B1	113
				178	B2	114
				179	B3	115
				180	B4	116
				181	B5	117
				182	B6	118
				183	B7	119
				184	B8	120
				185	B9	121
	✓	✓		186	BA	122
				187	BB	123
				188	BC	124
				189	BD	125
				190	BE	126
				191	BF	127
				192	C0	64

Caratteri Standard PET	Caratteri Standard CBM	Caratteri Alternativi PET	Caratteri Alternativi CBM	PET ASCII DEC	PET ASCII HEX	PEEK/POKE
♠	A	A	♠	193	C1	65
	B	B		194	C2	66
	C	C		195	C3	67
	D	D		196	C4	68
	E	E		197	C5	69
	F	F		198	C6	70
	G	G		199	C7	71
	H	H		200	C8	72
	I	I		201	C9	73
	J	J		202	CA	74
	K	K		203	CB	75
	L	L		204	CC	76
	M	M		205	CD	77
	N	N		206	CE	78
	O	O		207	CF	79
	P	P		208	D0	80
●	Q	Q	●	209	D1	81
	R	R		210	D2	82
♥	S	S	♥	211	D3	83
	T	T		212	D4	84
	U	U		213	D5	85
	V	V		214	D6	86
○	W	W	○	215	D7	87
♣	X	X	♣	216	D8	88
	Y	Y		217	D9	89
♦	Z	Z	♦	218	DA	90
+	+	+	+	219	DB	91
				220	DC	92
				221	DD	93
π			π	222	DE	94
				223	DF	95
				224	E0	96
				225	E1	97
				226	E2	98
				227	E3	99
				228	E4	100
				229	E5	101
				230	E6	102
				231	E7	103
				232	E8	104
				233	E9	105
				234		106
				235		107
				236		108
				237		109
				238		110
				239		111
				240		112
				241		113
				242		114
				243		115
				244		116
				245		117
				246		118
				247		119
				248		120
				249		121
	✓	✓		250		122
				251		123
				252		124
				253		125
				254		126
π			π	255		127

Tabella D-3. Caratteri del sistema PET/CBM

Caratteri Standard PET	Caratteri Standard CBM	Caratteri Alternativi PET	Caratteri Alternativi CBM	PET ASCII DEC	PET ASCII HEX	PEEK/POKE
						128
	a	a				129
	b	b				130
	c	c				131
	d	d				132
	e	e				133
	f	f				134
	g	g				135
	h	h				136
	i	i				137
	j	j				138
	k	k				139
	l	l				140
	m	m				141
	n	n				142
	o	o				143
	p	p				144
	q	q				145
	r	r				146
	s	s				147
	t	t				148
	u	u				149
	v	v				150
	w	w				151
	x	x				152
	y	y				153
	z	z				154
						155
						156
						157
						158
						159
						160
						161
						162
						163
						164
						165
						166
						167
						168
						169
						170
						171
						172
						173
						174
						175
						176
						177
						178
						179
						180
						181
						182
						183
						184
						185
						186
						187
						188
						189
						190
						191
						192
						193
						194
						195
						196
						197
						198
						199
						200
						201
						202
						203
						204
						205
						206
						207
						208
						209
						210
						211
						212
						213
						214
						215
						216
						217
						218
						219
						220
						221
						222
						223
						224
						225
						226
						227
						228
						229
						230
						231
						232
						233
						234
						235
						236
						237
						238
						239
						240
						241
						242
						243
						244
						245
						246
						247
						248
						249
						250
						251
						252
						253
						254
						255

re-verse

APPENDICE E

FUNZIONI DEFINITE DALL'UTENTE IN TERMINI DI FUNZIONI IMPLEMENTATE DAL BASIC

SECANTE	DEF FNA(X)=1/COS(X) valida per X diverso da Π/2
COSECANTE	DEF FNB(X)=1/SIN(X) valida per X diverso da 0
COTANGENTE	DEF FNC(X)=COS(X)/SIN(X) valida per X diverso da 0
ARCOSENO	DEF FND(X)=ATN(X)/SQR(—X*X+1)) valida per ABS(X) minore di 1
ARCOCOSENO	DEF FNE(X)=—ATN(X/SQR(—X*X+1))+Π/2 valida per ABS(X) minore di 1
ARCOSECANTE	DEF FNF(X)=ATN(SQR(X*X—1)+(SGN(X)—1)*Π/2 valida per ABS(X) maggiore 1
ARCOCOSECANTE	DEF FNG(X)=ATN(1/SQR(X*X—1))+(SGN(X)—1)*Π/2 valida per ABS(X) maggiore di 1
ARCOCOTANGENTE	DEF FNH(X)=—ATN(X)+Π/2 valida per qualunque X
SENOIPERBOLICO	DEF FNI(X)=(EXP)X — EXP(—X))/2 valida per qualunque X
COSENO-IPERBOLICO	DEF FNJ(X)=(EXP(X)+EXP(—X))/2 valida per qualunque X

TANGENTE-IPERBOLICA	DEF FNK(X) = EXP(—X)/(EXP(X)+EXP(—X))*2+1 valida per qualunque X
SECANTE-IPERBOLICA	DEF FNL(X)=2/(EXP(X)+EXP(—X)) valida per qualunque X
COSECANTE-IPERBOLICA	DEF FNM(X)=2/EXP(X)—EXP(—X)) valida per X diverso da 0
COTANGENTE-IPERBOLICA	DEF FNN(X) = EXP(—X)/(EXP(X)+EXP(—X))*2+1 valida per X diverso da 0
ARCOSENO-IPERBOLICO	DEF FNO(X)=LOG(X+SQR(X*X+1)) valida per qualunque X
ARCOCOSENO-IPERBOLICO	DEF FNP(X)=LOG(X*SQR(X*X —1)) valida per X maggiore o uguale 1
ARCOTANGENTE-IPERBOLICA	DEF FNQ(X)=LOG((1+X)/(1—X))/2 valida pe ABS(X) minore di 1
ARCOSECANTE-IPERBOLICA	DEF FNR(X)=LOG((SQR(—X*X+1)+1)/X) valida per X maggiore 0 e minore o uguale a 1
ARCOCOSECANTE-IPERBOLICA	DEF FNS(X)=LOG ((SGN(X)*SQR (X*X+1) 1)/X) valida per X diverso da 0
ARCO-COTANGENTE-IPERBOLICA	DEF FNT(X)=LOG((X+1)/(X—1))/2 valida per ABS(X) maggiore di 1

APPENDICE F

INTERFACCE E LINEE DEL PET

Nella figura F.1. è riportato uno schema semplificato dei collegamenti possibili con apparecchiature esterne. Nella figura F.2. è riportato uno schema semplificato dei connettori J1 e J2. Il connettore J1 non è uno standard IEEE-488, ma si usa un connettore con 24 punti di contatto con 0.156 inch di spazio tra i centri dei contatti stessi. Nella tabella a pagina 158 sono riportati i caratteri di identificazione dei contatti, le corrispondenze con lo standard IEEE-4888, il significato mnemonico e la definizione dei segnali.

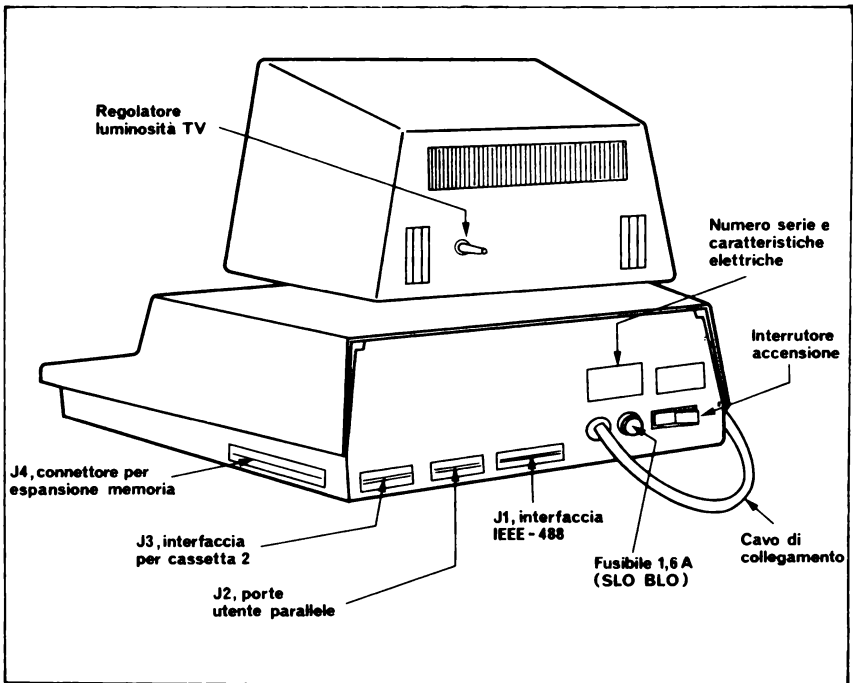



Figura F-1. Schema semplificato del retro del calcolatore.

Caratteri PIN CBM	Standard IEEE	Segnali IEEE mnemonici	Segnali definizioni
1	1	DI01	I/O dati linea 1
2	2	DI02	I/O dati linea 2
3	3	DI03	I/O dati linea 3
4	4	DI04	I/O dati linea 4
5	5	EOI	Fine identificazione
6	6	DAV	Dati validi
7	7	NRFD	Non pronto per dati
8	8	NDAC	Dati non accettati
9	9	IFC	Reset interfaccia
10	10	SRQ	Richiesta servizio
11	11	ATN	Segnale attention
12	12	GND	Massa del telaio
A	13	DI05	I/O dati linea 5
B	14	DI06	I/O dati linea 6
C	15	DI07	I/O dati linea 7
D	16	DI08	I/O dati linea 8
E	17	REN	remoto non pronto
F	18	GND	DAV MASSA
H	19	GND	NRFD MASSA
J	20	GND	NDAC MASSA
K	21	GND	IFC MASSA
L	22	GND	SRQ MASSA
M	23	GND	ATN MASSA
N	24	GND	(DI01-8) MASSA

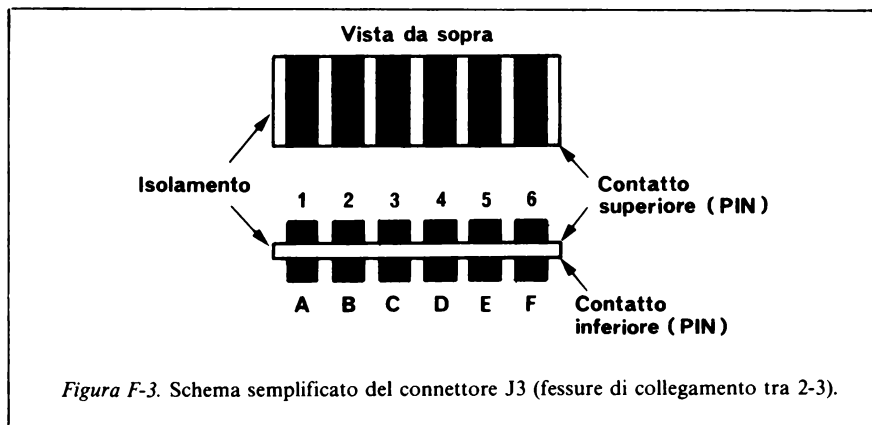
Nella tabella che segue sono riportate le caratteristiche per il connettore J2.

Caratteri PIN	Segnali mnemonici	Segnali definizioni
1	Ground	
2	TV Video	Uscita del video, usato per i test diagnostici del video
3	IEEE-SRQ	Collegamento diretto al segnale SRQ nelle porte IEEE-488. Usato nelle routine diagnostiche per verificare SRQ
4	IEEE-EOI	Collegamento diretto al segnale EOI nelle porte IEEE-488. Usato nelle routine diagnostiche.
5	Per uso routine diagnostiche	Se questo collegamento è mantenuto basso, durante l'uso il software va alla routine diagnostiche invece che al Basic.
6	Cassetta 1 lettura	Usato con le routine diagnostiche per verificare la lettura
7	Cassetta 2 lettura	Come sopra per cassetta 2.
8	Scrittura cassette	Per verificare la scrittura delle due cassette
9	TV Verticale	Per verificare i segnali verticali della TV. Può essere usato per TV esterne.
10	TV Orizzontale	Per verificare i segnali orizzontali della TV. Può essere usato per TV esterne
11,12	GND	Massa del segnale
A	GND	Massa del segnale
B	CA1	Contatto sensibile per input standard 6522VIA
C	PA0	Da PA0 a P17 sono di I/O alle
D	PA1	periferiche e possono essere
E	PA2	programmate indipendentemente
F	PA3	l'una dall'altra per input e output
H	PA4	
J	PA5	
K	PA6	
L	PA7	
M	CB2	Speciali PIN I/O di VIA
N	GND	Massa del segnale

Figura F-2. Schema semplificato dei connettori J1 e J2 (fessure di collegamento tra 2-3 e 9-10 per J1, tra 1-2 e 10-11 per J2).

I segnali CA1, PA0-7 e CB2 sono direttamene collegati allo standard 6522 VIA localizzato a partire dall'indirizzo esadecimale E840, decimale 59456.

Le porte parallele consistono in linee programmabili bidirezionali per I/O. All'indirizzo decimale 59459 (esadecimale E843) si trova il registro direzionale; i suoi 8 bits sono in corrispondenza con le 8 linee di I/O PA0-7. Un bit a zero nel registro direzionale (DDRA) significa che la linea corrispondente è usata per INPUT, mentre un bit a 1 significa che la linea corrispondente è usata come OUTPUT. Con la POKE si può configurare il registro DDRA nel modo voluto e poi scrivere per l'OUTPUT il dato voluto nel registro 59471 o leggere con la PEEK sempre da 59471 il dato ricevuto.

Figura F-3. Schema semplificato del connettore J3 (fessure di collegamento tra 2-3).

Esempio: con POKE 59459,15 si ottiene in 59459 00001111 e così si attivano come output le porte da PA0 a PA3, poi con POKE 59471,255 si mandano fuori 4 segnali solo dalle porte PA0-3, mentre le porte PA4-7 restano inalterate. Se si fa POKE 59471,0 i segnali diventano bassi sulle porte PA0-7 e le altre restano inalterate; se si fa POKE 59471,3 le porte A0-1 diventano con segnale alto, PA2-3 con segnale basso e le altre inalterate.
A seconda dei dispositivi collegati, potrà o meno essere possibile programmare in BASIC. In tutti i casi si potrà programmare in linguaggio macchina.

Nella figura F. 3. è riportato uno schema semplificato del connettore J3 per la seconda cassetta.

APPENDICE G

IL BUS DEL PET

Il BUS consiste in 24 linee divise funzionalmente in 3 gruppi:

1. linee per la trasmissione dei dati
2. linee per il controllo
3. linee per la direzione (governo)

secondo lo schema seguente.

Identific. contatti CBM	BUS	Nomi IEEE	Identific. contatti CBM	BUS	Nomi IEEE
1	DATI	DI01	A	DATI	DI05
2	DATI	DI02	B	DATI	DI06
3	DATI	DI03	C	DATI	DI07
4	DATI	DI04	D	DATI	DI08
5	GOVERNO	EOI	E	GOVERNO	REN
6	DATI	DAV	F	TERRA	GND6
7	DATI	NRFD	H	TERRA	GND7
8	DATI	NDAC	J	TERRA	GND8
9	GOVERNO	IFC	K	TERRA	GND9
10	DATI	SRQ	L	TERRA	GND10
11	DATI	ATN	M	TERRA	GND11
12	DATI	SHIELD	N	TERRA	LOGIC GND (DI01-8)

Il BUS ha i seguenti *utenti:*

1. Apparecchiature che trasmettono dati;
2. Apparecchiature che ricevono dati; possono essere attive contemporaneamente più apparecchiature.

3. L'unità di governo del calcolatore, che è il solo controllore delle attività del BUS.

Nei collegamenti al BUS si hanno le seguenti *limitazioni:*

1. Massima lunghezza del cavo 20 metri,
2. Massima distanza tra i dispositivi 5 metri,
3. Massimo numero di dispositivi collegati 15,
4. Massima velocità di trasferimento dei dati 250 kHz.

Linee per trasmissione dati

Sono le linee da DI01 e DI08. Sono linee bidirezionali con stato attivo basso, cioè sulle linee è normalmente attiva una tensione. La velocità di trasmissione è imposta dal dispositivo più lento. Vengono trasmessi 8 bits in parallelo e quindi si ha una trasmissione in serie di bytes. Il bit più significativo corrisponde a DI08. Lungo queste linee vengono trasmessi

- dati da o a periferiche,
- indirizzi,
- informazioni di controllo.

Gli indirizzi e le informazioni di controllo si distinguono dai dati per essere ATN ON durante il loro trasferimento. Gli indirizzi in realtà possono arrivare a 65535 e quindi avrebbero bisogno di 16 bits per essere trasmessi, ma il sistema usa la divisione della memoria in pagine e quindi trasmette con 8 bits la pagina e con gli 8 seguenti l'indirizzo nella pagina.

Linee per il controllo

Sono 3 e controllano il trasferimento dei dati. Esse sono: DAV, NRFD, NDAC.

DAV, dato valido. Quando si porta allo stato basso i ricevitori, possono leggere il byte sulle linee dati. Il trasmettitore non può portare allo stato basso la linea DAV finchè la NRFD è bassa, in quanto tutti i ricevitori devono essere pronti per accettare il nuovo dato.

NRFD, non pronto per i dati. Se è allo stato basso uno o più ricevitori non sono pronti ad accettare il prossimo dato. Quando tutti i dispositivi sono pronti NRFD va allo stato alto informando il trasmettitore che può essere caricato il byte da trasmettere.

NDAC, dato non accettato. Ciascun ricevitore la mantiene allo stato basso fino a quando non ha finito di leggere un dato. Quando NDAC va allo stato alto, il trasmettitore può caricare un nuovo dato sulle linee.

Nel diagramma che segue si dà la sequenza degli eventi durante una trasmissione di dati. Gli eventi sono numerati da 1 a 7, e dopo il diagramma segue la spiegazione.

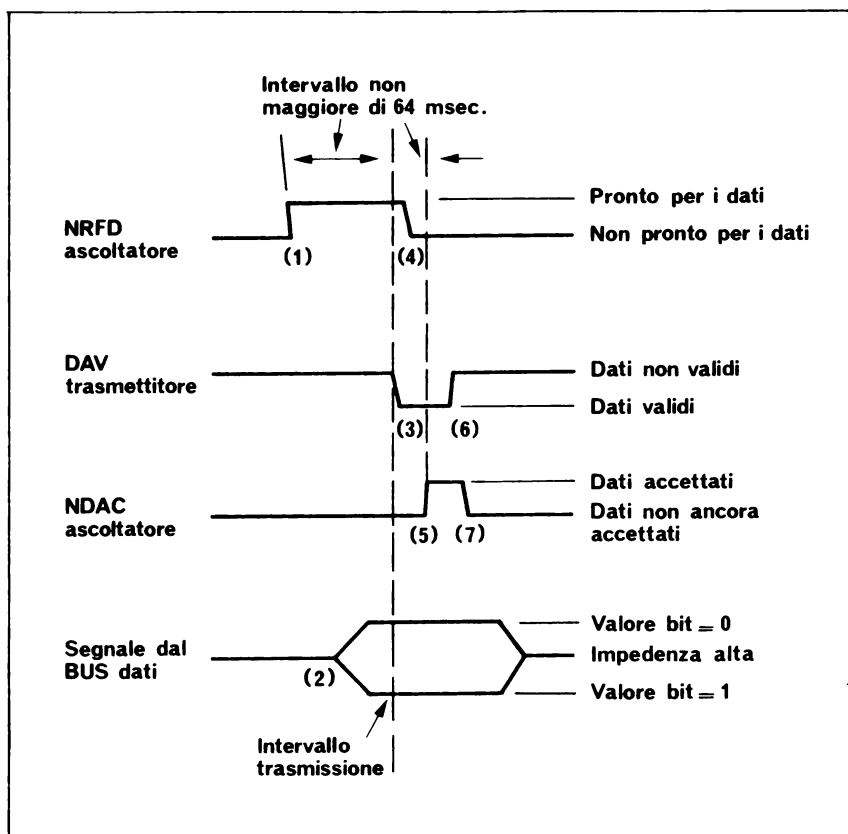



Figura G-1. Sequenza di trasmissione dati.

Sequenza di trasmissione dati:

(1) Quando NRFD va allo stato alto il trasmettitore può inviare un nuovo dato.
(2) Il trasmettitore invia il dato sulle linee dei dati dopo un breve intervallo.
(3) Il trasmettitore, quando trova NRFD alto mette DAV basso per informare che il dato è valido.
(4) Quando tutti hanno ricevuto il dato NDAC va allo stato alto.
(5) NDAC alto significa che il trasmettitore può eliminare il dato già trasmesso.
(6) Il trasmettitore pone DAV allo stato alto.
(7) Il ricevitore trovando DAV alto pone NDAC basso e poi NRFD viene rilasciato e può ricominciare il ciclo.

Linee di governo

Servono per gestire il sistema; esse sono: ATN, EOI, IFC, SRQ, REN.

ATN, segnale di attenzione. Viene messo allo stato basso (TRUE o 1 logico) quando il governo vuole assegnare alle periferiche il compito di essere ricevitori o trasmettitori. Se ATN è basso, nel BUS possono circolare solo indirizzi o messaggi di comando. Se ATN è alto, nel BUS circolano dati gestiti dai dispositivi predefiniti.

SRQ, richiesta di servizio. Viene posto dai dispositivi allo stato basso per richiedere servizio al governo. Allora il controllore pone basso ATN e ricerca quale dispositivo ha richiesto servizio. Non si può controllare in BASIC, ma in linguaggio macchina.

IFC, reset delle interfaccie. Il governo attiva questa linea per inizializzare il BUS. Il PET aziona questa linea quando viene acceso, per circa 100 msec.

REN, abilitazione remota. Alcuni dispositivi possono essere accesi sia con un interruttore in loco che con un comando attraverso il BUS. L'accensione può essere fatta mantenendo REN basso. Nel PET la linea è sempre bassa.

EOI, fine o identificazione. Quando un trasmettitore ha finito di inviare dati, può portare allo stato basso EOI. Il governo mette sempre EOI allo stato basso quando ha finito di trasmettere dati.

Segue una tabella degli indirizzi di memoria usati per il BUS.

Indirizzo esadecimale	Indirizzo decimale	Bits	IEEE	Modo
E820	59424	0-7	DI01-8	Input
E822	59426	0-7	DI01-8	Output
E821	59425	3	NDAC	Output
E823	59427	3	DAV	Output
		7	SRQ	Input
E810	59408	6	EOI	Input
E840	59456	0	NDAC	Input
		1	NRFD	Output
		2	ATN	Output
		6	NRFD	Input
		7	DAV	Input

APPENDICE H

FRASI BASIC

Nel riepilogare le frasi BASIC si usano le seguenti convenzioni:

- V e W denotano variabili numeriche
- X denota un'espressione numerica
- X$ denota una stringa
- I e J denotano un'espressione numerica troncata alla sua parte intera prima di eseguire la frase

DEF	100 DEF FNA(V)=V/B+C	A può essere qualunque nome di variabile, ma non di stringa. V è l'argomento fittizio (DUMMY), che serve solo per la definizione e può essere sostituito dall'argomento reale quando la funzione è richiamata. L'espressione da calcolare deve stare su una sola linea. Si possono definire funzioni in modo ricorsivo, il sistema usa lo Stack per gestirle: 200 DEF FNA(V)=FNB(V).
DIM	113 DIM A(3),B(10)	predispone spazio per variabili con indice, le matrici vengono azzerate.
	114 DIM R3(3,5),D$(2,2,2)	le variabili con indice (matrici) possono avere più di un indice. Praticamente i limiti sono dati dalla dimensione della memoria.
	115 DIM Q1(N),Z(2*1)	le MATRICI possono essere dimensionate in modo dinamico durante l'esecuzione.

Se non c'è il dimensionamento, la prima volta che viene chiamato un elemento della matrice, essa viene dimensionata con il numero di indici della chiamata e con ogni dimensione pari a 10 e quindi di 11 elementi.
Gli indici partono da zero e quindi per ogni indice si ha un elemento di più di quelli dichiarati.

END 999 END

fa terminare l'esecuzione del programma senza stampare messaggi. Se dopo si dà il CONT il programma prosegue dalla frase seguente.

FOR ... NEXT
300 FOR V=1 TO 9.3 STEP 16

V viene posto al valore dopo uguale, valore iniziale della variabile V di controllo del ciclo. Poi sono eseguite le frasi tra il FOR ed il NEXT. Il valore finale per V è quello che segue TO. Quando viene incontrato NEXT la variabile V viene incrementata del valore dopo STEP. Se V si mantiene minore o uguale al valore finale, l'esecuzione prosegue con la frase dopo FOR, se no l'esecuzione prosegue con la frase dopo il NEXT.
STEP non è obbligatorio, se manca, viene assunto uguale ad 1. Il ciclo FOR/NEXT viene eseguito almeno una volta anche se il valore di partenza di V è in contrasto con il valore finale.
I valori dopo =,TO e STEEP possono anche essere espres-

		sioni, comprendenti variabili. Queste espressioni vengono calcolate una volta sola all'inizio del FOR e poi adoperate. NEXT può essere seguito da V, così: NEXT V, ma può anche essere solo: NEXT ed allora indica la chiusura dell'ultimo FOR. Si può scrivere NEXT V,W per chiudere due FOR.
GET	GET A, GET # A GET A$, GET # A$	riceve un carattere dalla periferica alla quale si riferisce. Se si riferisce alla tastiera non aspetta il CR per accettare il carattere e così si avrebbe A = 0 o A$="", stringa nulla.
	10 GET A$: IF A$=""THEN 10	crea un ciclo di attesa fino a quando non si preme un tasto.
GOSUB RETURN	10 GOSUB 980	salta alla linea 980 e quando incontra un RETURN ritorna alla linea seguente la 10. Si possono nidificare i GOSUB fino a 23 volte. Dato che la ricerca dei sottoprogrammi parte dall'inizio del programma, conviene mettere i sottoprogrammi in testa al programma.
GOTO	50 GOTO 300	fa saltare alla linea indicata.
IF ... GOTO	10 IF X=Y GOTO 90	se la condizione è vera va alla linea 90.
IF ... THEN	15 IF X=Y THEN Y=Y+6 20 IF X=Y THEN 300	se la condizione è vera esegue le istruzioni dopo il THEN o va al numero di linea indicato. Dopo il THEN si possono scrivere più istruzioni separandole con i due punti.

INPUT	3 INPUT V,W	richiede dati dalla tastiera, se si scrive INPUT # allora richiede dati dalla periferica a cui si riferisce. I dati devono essere forniti separati da virgola. Se si danno meno dati stampa al video??. Se si danno più dati di quelli richiesti, dà un avviso EXTRA IGNORED e prosegue. Un comando di INPUT è interrotto dal tasto CR usato da solo. Se si scrive CONT viene rieseguito il comando INPUT interrotto. Salta gli spazi prima del primo carattere significativo
	10 INPUT "SCRIVI":V	fa scrivere la stringa tra apici prima del ? di richiesta dati.
LET	500 LET X=45 500 X=45	assegna valore ad una variabile; LET è opzionale.
ON ... GOTO	10 ON I GOTO 10,20,30	usa I come puntatore ai numeri di linea che seguono GOTO, se i=1 va a 10, se I=2 va a 20, se I=3 va a 30. Se I=0 oppure supera il numero di numeri di linea elencati, l'esecuzione prosegue con la frase seguente. La frase non può superare 79 bytes. Al posto di I si può usare un'espressione aritmetica.
ON ... GOSUB	10 ON I GOSUB 10,20	lo stesso di prima salvo che va ad eseguire dei sottoprogrammi.
POKE	10 POKE I,J	scrive nel byte I il valore J. J deve essere maggiore o uguale a zero e minore o uguale a 255. I deve essere positivo e compreso tra 0 e 65535.

PEEK	10 A=PEEK(I)	pone in A il valore del byte di indirizzo I.
PRINT	10 PRINT X,Y 10 PRINT X;Y 10 PRINT A$	fa uscire sul video il contenuto delle variabili che seguono PRINT. Se la lista di variabili termina con, o; non va a capo.
	10 PRINT "PIOVE"A$	se le variabili sono separate da ; non spazia, se sono separate da , spazia. Se le stringhe non sono separate da alcun separatore, le evidenzia una vicina all'altra. PRINT # opera sulle altre periferiche.
READ/DATA		
	10 READ V,W 5 DATA 12,5,7	legge nelle variabili V e W due dati dal blocco dati definito da DATA, e fa avanzare il puntatore interno nel blocco di dati. Se si leggono più dati di quanti disponibili si ha l'avviso OUT OF DATA.
RESTORE	50 RESTORE	riposiziona il puntatore interno all'inizio del blocco dati.
REM	50 REM commenti	serve per introdurre commenti in un programma dopo una frase REM non sono validi i ":"
STOP	80 STOP	fa fermare il programma con il messaggio BREAK IN LINE 80. Se si usa CONT si prosegue dalla linea seguente.
CONT		può essere usato solo in modo diretto e fa proseguire l'esecuzione dopo STOP, END.
WAIT	WAIT I,J,K	crea un'attesa nel programma fino a quando una locazione I di memoria diventa divesa da zero. Opera così: 1. preleva il contenuto del byte I

2. opera un OR esclusivo (XOR) del contenuto di I e del contenuto di K, che fa da maschera. Ricordiamo le regole dell'operazione XOR:

   1 XOR 1 = 0
   0 XOR 1 = 1
   1 XOR 0 = 1
   0 XOR 0 = 0.

   Se manca esso è assunto uguale a zero.
3. opera un AND tra il valore ottenuto in 2) e la maschera J.
4. se il risultato è 0, wait ritorna su se stesso e ricomincia dal punto 1.
5. se il risultato è diverso da 0 l'esecuzione prosegue con la frase seguente WAIT.

Durante un WAIT non viene recepito il tasto STOP.

APPENDICE I

I COMANDI BASIC

I comandi BASIC vengono di norma usati dopo che sullo schermo compare READY, cioè quando il calcolatore si trova in "Livello di Comando". I comandi possono anche essere usati in un programma; alcuni, come LIST e NEW, quando vengono eseguiti fanno ovviamente terminare l'esecuzione del programma nel quale sono contenuti.

CLR		cancella tutte le variabili ed i riferimenti memorizzati.
LIST	LIST X	lista la linea X del programma,
	LIST o LIST —	lista tutto il programma, la parte che entra nello schermo, o tutto sulla stampante se è attivata,
	LIST X—	lista dalla linea X alla fine,
	LIST —X	lista fino alla linea X.
	LIST Y—X	lista il programma dalla linea Y alla linea X.
LOAD	LOAD	carica il primo programma che trova sulla cassetta 1.
	LOAD "nome"	cerca sulla cassetta 1 il programma "nome" e lo carica
	LOAD "nome", 2	come prima, ma dalla cassetta 2.
	LOAD "nome", n	carica il programma dalla periferica, avente numero n. Se il comando LOAD è inserito in un programma, l'esecuzione del programma in corso si ferma, viene caricato

		il nuovo programma. Il nuovo programma va in esecuzione dalla sua prima istruzione. Le variabili del vecchio programma sono passate al nuovo, purchè esso non sia più lungo del vecchio.
NEW	NEW	cancella il programma in uso e tutte le variabili.
RUN	RUN	cancella tutte le variabili e fa il RESTORE del blocco Dati e fa partire l'esecuzione del programma dalla prima istruzione. Se si desidera partire da un numero di linea e non cancellare le variabili, si deve usare GOTO n.
	RUN n	come prima, ma l'esecuzione parte da n.
SAVE	SAVE	memorizza il programma in uso sulla cassetta 1.
	SAVE "nome"	memorizza il programma in uso sulla cassetta 1 con il nome "nome".
	SAVE "nome",n	memorizza il programma "nome" sul dispositivo n.
	SAVE "nome", 2,1	memorizza il programma "nome" sulla cassetta 2 e scrive il blocco di fine nastro.
VERIFY	VERIFY "nome"	confronta il contenuto della memoria con il programma "nome", valgono i parametri per i dispositivi.

APPENDICE L

FUNZIONI, ESPRESSIONI E OPERATORI

Funzioni Matematiche		
Funzione	**Esempio**	**Commento**
ABS	10 C=ABS(A)	fornisce il valore assoluto di A
ATN	10 C=ATN(A)	fornisce l'arcotangente di A in radianti
COS	10 C=COS(A)	fornisce il coseno di A; A deve essere in radianti
DEF FN	10 DEF FNA(B)=B+ D/E	permette al programmatore di definire una funzione; il nome della funzione deve essere un nome di variabile numerica, l'argomento B è solo esplicativo (DUMMY).
EXP	10 C=EXP(A)	fornisce e^A (e elevato a A)
INT	10 C=INT(A)	fornisce il numero intero più grande minore o uguale ad A
LOG	10 C=LOG(A)	fornisce il logaritmo naturale di A; A deve essere maggiore o uguale a zero
RND	10 C=RND(A)	fornisce un numero a caso tra 0 e 1. Se A è negativo viene fornito lo stesso numero a caso ad ogni esecuzione. Se A=0 si genera la stessa sequenza di numeri. Se A è maggiore di zero, si produce una nuova sequenza di numeri a caso ad ogni chiamata
SGN	10 C=SGN(A)	fornisce —1 se A è negativo, 0 se A=0, +1 se A è positivo
SIN	10 C=SIN(A)	fornice il seno di A; A deve essere in radianti
SQR	10 C=SQR(A)	fornisce la radice quadrata di A
TAN	10 C=TAN(A)	fornisce la tangente di A; A deve essere in radianti

Funzioni di stringa		
Funzione	**Esempio**	**Commento**
ASC	10 A=ASC("XYZ")	fornisce il valore intero corrispondente al codice ASCII del primo carattere della stringa argomento, che può anche essere una variabile
CHR$	10 A$=CHR$(N)	fornisce il carattere corrispondente al codice ASCII N
LEFT$	10 ?LEFT$(X$,A)	fornisce agli A caratteri più a sinistra della stringa
LEN	10 ?LEN(X$)	fornisce la lunghezza della stringa
MID$	10 ?MID$(X$,A,B)	fornisce B caratteri della stringa partendo dalla posizione A
RIGHT$	10 ?RIGHT$(X$,A)	fornice gli A caratteri più a destra della stringa
STR$	10 A$=STR$(A)	fornisce la rappresentazione in forma di stringa di un numero A preceduto da spazio o da —.
VAL	10 A=VAL(A$)	fornisce la rappresentazione numerica della stringa A$; se la stringa non è numerica, fornisce 0.

NOTA. Le funzioni di stringa che forniscono valori numerici possono essere usate nelle espressioni numeriche.
Negli esempi si sono usate solo frasi di assegnazione e di stampa, ovviamente si possono usare anche altre frasi BASIC.

Operatori aritmetici

= assegna un valore alla variabile scritta a sinistra,
— se usato prima di un numero o di una variabile ha il significato di negazione, mentre se usato tra due variabili ha il significato di sottrazione,
↑ significa "elevato a". 0 ↑ 0 dà 1; 0 ↑ n dà 0 per qualunque n; A ↑ B dà errore di tipo FC se A è negativo e B non intero,
* significa moltiplicazione,
/ significa moltiplicazione,
+ significa addizione,
— significa sottrazione.

Operatori relazionali

=   uguale,
<>   non uguale,
>   maggiore,
<   minore,
<=, =< minore o uguale,
>=, => maggiore o uguale.

NOTA. Gli operatori relazionali possono essere usati all'interno di espressioni aritmetiche; si tenga presente che la relazione vera dà luogo al valore – 1 e la relazione falsa al valore 0. Cioè 5 = 4 dà 0, mentre 5 = 5 dà –1. Se si scrive IF X THEN ... è come se si scrivesse IF X<>0 THEN, cioè esegue le istruzioni dopo THEN se X è diverso da zero.

Operatori Booleani

AND   significa l'uno e l'altro
OR   significa l'uno o l'altro
NOT   significa la negazione.

Gli operatori booleani possono essere usati per legare insieme espressioni relazionali ed allora hanno il seguente significato:

Espressione —1 AND espressione —2 è vero se sia espressione —1 che espressione —2 sono vere.
Espessione —1 OR espressione —2 è vero se almeno una delle due espressioni è vera.
NOT espressione —1 è vero se espressione —1 è falsa.

Inoltre questi operatori possono essere usati per manipolare i bit nella memoria, tenendo conto delle seguenti regole:

1 AND 1 = 1   1 OR 1 = 1
0 AND 1 = 0   1 OR 0 = 1
1 AND 0 = 0   0 OR 1 = 1
0 AND 0 = 0   0 OR 0 = 0

NOT 1 = 0
NOT 0 = 1

Quando si usano gli operatori booleani, i loro argomenti vengono estesi a valori a due bytes e quindi compresi tra — 32768 e + 32767 ed essi operano secondo le regole su esposte.

63 AND 16 = 16 infatti 63= 0000 0000 0011 1111
                       16 = 0000 0000 0001 0000

---

risultato 16 = 0000 0000 0001 0000

come si vede usando l'operatore AND si possono fabbricare delle maschere (opportuni insieme di 1 e 0) per estrarre parti di variabili, infatti vengono mantenuti i bits che corrispondono agli 1 della maschera.

143 OR 272 = 415   infatti 143 = 0000 0000 1000 1111
                    272 = 0000 0001 0001 0000

                    risultato 415 = 0000 0001 1001 1111

NOT 1 = —2   infatti 1 = 0000 0000 0000 0001 cambiando gli zero in 1 e viceversa si ottiene 1111 1111 1111 1110 che è uguale a —2 (si ricordi che i numeri negativi sono rappresentati in forma complementata)

Espressioni

Tenendo conto che in un'espressione vengono eseguite per prime le operazioni contenute nelle parentesi più interne, segue l'ordine di precedenza degli operatori. Se i due operatori hanno lo stesso ordine di precedenza, viene eseguita prima l'operazione indicata più a sinistra.

1. elevamento a potenza
2. — come negazione, cioè opeatore unitario
3. */ moltiplicazione e divisione
4. + — somma e sottrazione
5. operatori relazionali tutti con la stessa precedenza: = < > < > <= >=
6. NOT nel senso di negazione logica, manipolazione dei bits
7. AND
8. OR

INDICE ANALITICO

L. 11.500

Cod. 506 B

La dr. Rita Bonelli, laureata in Matematica e Fisica presso l'Università di Milano, può vantare un'esperienza di circa 25 anni nell'analisi dei sistemi organizzativi e nella programmazione dei calcolatori elettronici.
Iniziata la sua attività come venditore, ha poi lavorato sino al 1960, come analista-programmatore, sul primo calcolatore elettronico IBM installato in Italia presso un'industria.
Nei successivi 20 anni ha lavorato su medi e grandi sistemi elettronici della seconda e della terza generazione, svolgendo una intensa attività di consulenza professionale per fabbricanti di calcolatori, grandi aziende industriali e case di software.
Da più di dieci anni affianca alle attività professionali le attività didattiche, come titolare di una cattedra di informatica presso l'Istituto Tecnico Industriale Feltrinelli di Milano.
Attualmente si interessa anche di mini e personal computers, studiando il software applicativo per particolari categorie di utenti, e tenendo corsi di programmazione a vari livelli.

41

IMPARIAMO A PROGRAMMARE IN BASIC CON IL PET™/CBM™

R. Bonelli

GRUPPO EDITORIALE JACKSON